UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Desarrollo de un dispositivo IoT para la adquisición de medidas ambientales

Javier García Fernández
Tutor: Lluís Gifre Renom
Ponente: Jorge E. López de Vergara Méndez

Septiembre 2019

Desarrollo de un dispositivo IoT para la adquisición de medidas ambientales

AUTOR: Javier García Fernández
TUTOR: Lluís Gifre Renom

Departamento de Tecnología Electrónica y de las Comunicaciones
Grupo de la EPS: Computación y Redes de altas prestaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Septiembre 2019

Resumen (castellano)

Internet de las Cosas (Internet of Things, IoT) tiene como objetivo principal poder conectar todos los dispositivos a Internet. En el caso de dispositivos cotidianos, el fin será mejorar la calidad de vida del usuario. Por el contrario, cuando los dispositivos o maquinaria sean empresariales, el objetivo, normalmente, será mejorar los procesos de negocio.

Los dispositivos IoT normalmente se componen de sensores y actuadores combinados con módulos de comunicaciones para contactar con aplicaciones en la nube. Estos sensores, si existen, se emplean para tomar medidas del entorno y subirlas a la aplicación en la nube. La aplicación que maneja dichos dispositivos procesa las medidas o toma decisiones en base a otras fuentes de datos y, si el dispositivo dispone de actuadores, puede bajar acciones o comandos que el dispositivo tiene que desencadenar sobre sus actuadores.

Aquí surge la necesidad de encontrar un protocolo ligero que sea idóneo para manejar muchos dispositivos en paralelo, los cuales van a generar grandes cantidades de información que se actualiza con el tiempo. Este protocolo de mensajería debe emplear intermediarios de mensajería (como el bróker de mensajería Mosquitto), con el fin de permitir un intercambio de mensajes con dispositivos cuya conexión a la red no se puede garantizar ni ser continua y ni estable.

La finalidad de este Trabajo Final de Grado es el desarrollo de un sistema IoT capaz de recolectar medidas ambientales mediante sensores instalados en un dispositivo y su envío a una aplicación en la nube que pueda almacenarlas y representarlas en un cuadro de mandos. El problema es que para esto se necesita disponer de sistemas que puedan recibir medidas de distintos tipos de sensores además de traducir dichos mensajes a un lenguaje común, procesarlos y subirlos a un sistema central.

Para ello se ha escogido un protocolo de mensajería que, mediante la utilización de un gestor de mensajes, consiga que las muestras tomadas en un dispositivo se publiquen en un tópico donde la aplicación tenga acceso y pueda recibirlas. El protocolo MQTT (Message Queuing Telemetry Transport) hace de gestor para que los distintos dispositivos puedan publicar y escuchar en sus tópicos sin necesidad de que tengan o garanticen una conexión continua a la red. El broker de mensajería que se va a utilizar es Mosquitto, una solución de código abierto comúnmente usada en aplicaciones IoT y que soporta MQTT.

Finalmente, el almacenamiento de la información obtenida, así como la representación de una forma gráfica y actualizable de la misma se ha conseguido con las herramientas InfluxDB y Grafana. La primera para el almacenamiento de los datos generados y la segunda para la representación de los mismos en gráficas.

Abstract (English)

Internet of Things (IoT) has as main objective to connect all devices to the Internet. In the case of everyday devices, the purpose will be to improve the quality of life of the user. On the contrary, when the devices or machinery are business, the objective will normally be to improve business processes.

IoT devices typically consist of sensors and actuators combined with communication modules to connect with applications in the cloud. These sensors, if they exist, are used to take measurements of the environment and upload them to the application in the cloud. The application that manages these devices processes the measurements or makes decisions based on other data sources and, if the device has actuators, it can download actions or commands that the device has to trigger on its actuators.

Here arises the need to find a lightweight protocol that is suitable to handle many devices in parallel, which will generate large amounts of information that is updated over time. This messaging protocol must use messaging intermediaries (such as the Mosquitto messaging broker), in order to allow an exchange of messages with devices whose connection to the network cannot be guaranteed or be continuous and stable.

The purpose of this Final Degree Project is the development of an IoT system capable of collecting environmental measurements by means of sensors installed in a device and sending them to an application in the cloud that can store them and represent them in a scorecard. The problem is that for this you need to have systems that can receive measurements of different types of sensors as well as translate these messages into a common language, process them and upload them to a central system.

For this purpose, a messaging protocol has been chosen that, through the use of a message manager, ensures that samples taken on a device are published on a topic where the application has access and can receive them. The MQTT (Message Queuing Telemetry Transport) protocol acts as a manager so that different devices can publish and listen on their topics without having to guarantee a continuous connection to the network. The messaging broker that will be used is Mosquitto, an open source solution commonly used in IoT applications and that supports MQTT.

Finally, the storage of the information obtained, as well as the representation of a graphic and updatable form of it has been achieved with the InfluxDB and Grafana tools. The first for the storage of the generated data and the second for the representation of the same in graphs.

Palabras clave (castellano)

Internet de las cosas, gestor de mensajería, medidas ambientales, dispositivos IoT, cuadro de mandos IoT.

Keywords (inglés)

Internet of things, message broker, environmental measures, IoT devices, IoT dashboard.

Agradecimientos

En primer lugar, me gustaría dar gracias a todos los profesores que me han formado. En especial, a los que han sido más cercanos que hacían que la realización alumno-profesor fuese diferente.

Sobre todo, me gustaría dar gracias a mi tutor, Lluís, porque me ha ayudado siempre y sin él no habría sido posible haber acabado este proyecto. Cada vez que necesitaba ayuda me echaba una mano aunque apenas tuviese tiempo para hacerlo o me resolvía cualquier duda fuese fin de semana o altas horas de la noche. Ha sido un gran apoyo en esta última etapa y un gran profesor para cualquier alumno.

A todos mis amigos que me han acompañado durante toda la carrera, que cada mañana cuando llegara a clase hacían que mis ganas de afrontar el día a día nunca se agotaran. Sin todas esas sonrisas y buenos momentos hubiera sido una meta bastante difícil de conseguir. Porque sin todos los buenos y malos momentos que hemos pasado en las clases, laboratorios o en la cafetería no habría conseguido cerrar esta etapa.

Me gustaría dar gracias a mi pareja, Sandra. Sus ánimos día a día, su paciencia para contarle mis preocupaciones y los momentos que hemos pasado juntos han hecho de esta carrera un camino más ameno y fácil de afrontar.

Finalmente, dar las gracias a mi familia, que siempre me han apoyado y han apostado en todo lo que he hecho. Porque siempre han estado ahí con sus mejores caras cuando tenía un mal día y lo pagaba con ellos, cuando tenía un problema y se olvidaban de los suyos para animarme.

Muchas gracias a todos.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

En esta sección se va a explicar la motivación el trabajo, así como los objetivos que se pretenden alcanzar y su organización.

1.1 Motivación

Internet ha evolucionado en los últimos años de una forma exponencial, de tal forma que no solo se usa para permitir que las personas accedan a la información e interactúen entre ellas, sino que las máquinas entre ellas. Este nuevo paradigma es lo que se conoce como Internet de las Cosas (Internet of Things, IoT) [1].

Los dispositivos que se basan en este paradigma son llamados dispositivos IoT. Este tipo de dispositivos obtienen, mediante el uso de sensores, la información necesaria acerca de su entorno y la envían a un sistema encargado de procesarla para llevar a cabo los cambios en su configuración y poder realizar las acciones para las que haya sido programado.

Un claro ejemplo de este proceso sería un campo de riego automatizado, donde los dispositivos IoT recopilan las medidas ambientales del terreno y las envía a otro sistema que decida cuándo y con cuanta cantidad de agua regar el campo. Por esto mismo, surge la necesidad de crear un software que pueda recopilar las medidas ambientales necesarias de una forma configurable lo más genérica posible. Esto requiere desplegar en el campo varios dispositivos IoT, con dicho software, equipados con sensores ambientales como un sensor de luz ambiental, humedad o temperatura, y que periódicamente se activen, tomen medidas y las manden a una aplicación IoT en la nube que las recolecte, almacene y procese. Esta aplicación, en base a la información procesada, desencadenaría acciones de control de sobre el sistema de riego del campo

De esta manera el desarrollo de un software de estas características que se aplique a un dispositivo, que pueda trabajar con distinto tipos de sensores para conseguir las medidas del entorno, podría suponer grandes ventajas en el uso de tecnologías que basan su funcionamiento en Internet de las Cosas.

Además, para el funcionamiento de estos dispositivos IoT será necesario encontrar un protocolo de mensajería ideal para este tipo de comunicaciones. Los dispositivos IoT tienen la capacidad de conectarse a la red y comunicarse entre ellos, pero estos dispositivos no pueden garantizar una conexión estable y continua a la red. Por este motivo se deberá escoger un protocolo que no necesite de una conexión continua, como por ejemplo MQTT. Este protocolo se basa en el uso de un gestor de mensajería sobre el que interactúan los dispositivos de manera que publican mensajes en las colas que ofrece el gestor para que otros dispositivos accedan a ellas y puedan recibir el mensaje.

1.2 Objetivos

El principal objetivo ha sido el desarrollo de un sistema basado en Internet de las Cosas Este sistema tiene que ser capaz de tomar medidas ambientales y enviar dichas muestras a un sistema central. Este trabajo se ha centrado en la parte de adquisición de medidas y queda fuera de su ámbito el aplicar reconfiguraciones, como sería el actuar sobre el sistema de riego

Dado que el sistema está basado en el paradigma de IoT se ha llevado a cabo el uso de un gestor de mensajería mediante Mosquitto y MQTT [2]. El último objetivo fue conseguir

que los datos obtenidos fuesen visualizados de forma dinámica y personalizable por el usuario en un cuadro de mandos. Para ello, se ha usado:

i) **Docker:** Sistema de contenedores que empaqueta las aplicaciones junto con sus librerías para facilitar su despliegue y gestión [3].

ii) **InfluxBD:** Sistema de almacenamiento basado en series temporales de datos [4].

iii) **Grafana:** Herramienta Web que permite conectar con una base de datos y diseñar diferentes paneles para la visualización de datos [5].

1.3 Fases del Proyecto

Las fases del proyecto quedan representadas en la ***Figura 1-1*** mediante un diagrama de Gantt. Los detalles de las tareas recogidas en el diagrama se detallan en la ***Tabla 1-1***. La duración de las tareas está relacionada con el tiempo que se ha dedicado a abordar cada una de ellas. Esta repartición del tiempo se basa en un total de 300 horas dedicadas al proyecto entero, es decir, en un entorno laboral realista, equivaldría a un total de 7,5 semanas laborales asumiendo una jornada laboral de 8 horas.

La primera tarea del trabajo ha consistido en un estudio de los conceptos previos para el desarrollo. La segunda tarea engloba el desarrollo de los aplicativos de software necesarios para la aplicación y el dispositivo, En la tercera tarea se han realizado las pruebas del funcionamiento del sistema y, con ellas, la aplicación de correcciones necesarias. También se han verificado y cotejado los resultados obtenidos comparándolos con sensores externos al sistema. La última fase ha sido dedicada a la elaboración de la memoria y de la presentación, además de la preparación de la misma.

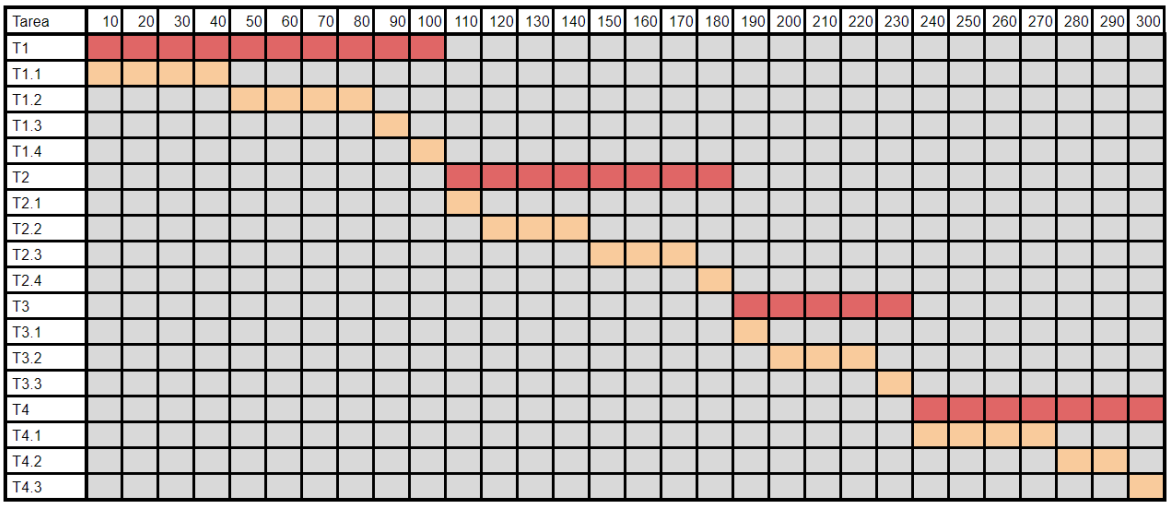

Figura 1-1: Diagrama de Gantt del Proyecto

Tabla 1-1: Fases del Proyecto

TAREAS	ACTIVIDADES	DURACIÓN (Horas)
T 1	Conocimientos previos	100
T 1.1	Estudio de Python/MicroPython	40
T.2	Estudio de protocolo MQTT	40
T.3	Estudio de dispositivos IoT	10
T 1.4	Estudio de Docker, InfluxDB y Grafana	10
T 2	Desarrollo del sistema	80
T 2.1	Diseño y desarrollo del protocolo de interacción entre dispositivo y aplicación	10
T 2.2	Implementación del software para la aplicación	30
T 2.3	Implementación del software para el dispositivo	30
T2.4	Extensión de la aplicación para el almacenamiento y visualización de muestras recibidas	10
T 3	Pruebas y resultados	50
T 3.1	Pruebas	10
T 3.2	Correcciones y mejoras	30
T 3.3	Valoración de resultados	10
T 4	Memoria	70
T 4.1	Diseño de la memoria	40
T 4.2	Elaboración de la presentación	20
T 4.3	Preparación de la defensa	10

1.4 Organización de la memoria

La memoria se ha organizado en los siguientes capítulos:

El capítulo 0, titulado *Estado del Arte*, se presentan los trabajos previos y tecnologías que se han usado durante el desarrollo del trabajo.

En el capítulo 3, titulado *Diseño y desarrollo*, se explica el planteamiento y funcionamiento del sistema desarrollado. Se mostrarán tanto la arquitectura física del sistema como la arquitectura software del mismo, así como la especificación del protocolo desarrollado.

En el capítulo 4, titulado *Integración, pruebas y resultados* se muestra el funcionamiento completo del sistema de tal manera que se comprueba mediante puntos de control que todo se ha realizado correctamente. Además, se muestran los resultados obtenidos en el transcurso de la ejecución.

Finalmente, en el capítulo 5, titulado *Conclusiones y trabajo futuro,* se resumen las conclusiones de este trabajo, valida el alcance de objetivos propuestos y propone líneas de desarrollo futuras.

2 Estado del arte

En este capítulo se van a explicar las tecnologías empleadas a lo largo del trabajo final de grado. Para ello se expondrá el concepto de dispositivo IoT, el protocolo de mensajería empleado y los aplicativos requeridos para el despliegue de la aplicación IoT.

2.1 Internet de las Cosas

En la actualidad, la tendencia es redefinir todos los dispositivos para convertirlos en dispositivos inteligentes. Estos dispositivos inteligentes, además de contener un chip microcontrolador, tienen la capacidad de conectarse a Internet para que puedan interactuar entre ellos. Como resultado, surge un nuevo uso de Internet que permite interconectar dispositivos o máquinas entre ellas dando lugar a lo que se conoce como Internet de las Cosas.

Los dispositivos IoT [1][6] pretenden mejorar los dispositivos clásicos incorporándoles capacidades de procesado, almacenamiento y conectividad. Estos dispositivos están dotados de sensores que toman medidas de su entorno y, con la información que obtienen, pueden controlar diferentes actuadores, o interactuar con una aplicación IoT en la nube que los gestione.

El uso de tecnologías IoT en dispositivos proporciona una serie de ventajas que los diferencian de sus versiones:

- **Seguimiento y monitorización de datos de usuarios, animales u objetos.** Localización del usuario mediante GPS, contabilización de la distancia recorrida, ritmo cardiaco. Por ejemplo los relojes inteligentes para deportistas.
- **Mayor independencia.** IoT proporciona una cierta autonomía a un sistema para realizar una tarea. Gracias a la interconexión de diferentes dispositivos IoT que se interconectan e intercambian información entre ellos, pueden automatizar ciertas tareas como el control de un sistema de riego inteligente o la temperatura de una casa inteligente sin intervención humana.
- **Seguridad.** Gracias a la capacidad de analizar el entorno y predecir situaciones indeseadas, pueden aportar seguridad en aplicaciones como la conducción autónoma de vehículos o la video-vigilancia del hogar y simulación de presencia para evitar robos cuando no estamos en casa.
- **Ahorro de tiempo y dinero.** Con aplicaciones IoT podremos ser capaces de hacer varias cosas simultáneamente y en otro lugar, lo que provoca un aprovechamiento del tiempo mayor. O el ahorro de dinero como por ejemplo un sistema de hogar inteligente que reduzca el consumo de luz y gas.

Sin embargo también existen un conjunto de inconvenientes que pueden hacer no tan eficiente o seguro el uso de IoT en nuestros dispositivos. Dado que los sistemas basados en IoT pueden tratar o estar relacionados con datos sensibles, ya sean de carácter personal o empresarial, es necesario prestar especial atención a aspectos como la seguridad y privacidad de estos. Una aplicación IoT médica o un sistema de control de una planta de fabricación inteligente comprometidos por un atacante podrían llegar a poner en riesgo la vida de los seres humanos o los trabajos que les permiten subsistir.

2.2 Dispositivos IoT

Tal como se ha introducido en la sección 2.1, un dispositivo IoT es básicamente un dispositivo o aparato clásico adaptado al paradigma de Internet de las Cosas, por tanto, ambas versiones del dispositivo podrán compartir una serie de características como sus componentes básicos de operación.

Los dispositivos IoT están formados, normalmente, por un sistema en un chip (System on Chip, SoC) que consiste en un circuito integrado dotado con un procesador o microcontrolador, memoria y conectividad. Gracias a la adición del SoC, un dispositivo común se puede transformar en un aparato capaz de realizar tareas de forma autónoma.

Los lenguajes de programación que aplican sobre este tipo de dispositivos son muy variados. Uno de los lenguajes más usados en placas microcontroladoras es el MicroPython, ya que es un lenguaje desarrollado cuya finalidad es el empleo en tecnologías IoT. Por esta razón MicroPython ha sido el lenguaje de programación escogido para la elaboración del trabajo. Las placas SoC de Pycom se programan en MicroPython. MicroPython [7] es un lenguaje de programación derivado de Python3 y específicamente orientado y optimizado a la programación de microcontroladores. Este lenguaje proporciona una selección de bibliotecas fundamentales de Python y además incluye módulos que permiten al programador el acceso al microcontrolador a bajo nivel, como por ejemplo, a los puertos de entrada/salida de propósito general (General Purpose Input / Output, GPIO) o a las interfaces digitales.

El SoC normalmente se conectará a una placa de aplicación que albergará los sensores y/o actuadores requeridos para la aplicación concreta a la que se destine el dispositivo. Normalmente, la interacción entre el SoC y los sensores y/o actuadores se llevará a cabo empleando diferentes tipos de interfaces dependiendo del tipo de datos y señales que se van a intercambiar.

Así, podemos tener dos tipos principales de interfaces: *i*) *analógicas*, cuando las señales que se intercambien entre el SoC y los sensores y/o actuadores sean de tipo analógico; y *ii*) *digitales*, cuando las señales intercambiadas sean digitales. En cuanto a las interfaces digitales, las más comunes son las interfaces SPI (Serial Peripheral Interface) e I2C(Inter-Integrated Circuit).

Las interfaces SPI e I2C son protocolos de comunicación que trabajan con buses de datos, cuyo objetivo principal es enviar y recibir datos desde la CPU hasta el resto de módulos del sistema. SPI es un protocolo de comunicación síncrono que trabaja con buses de datos, el cual ofrece una velocidad mayor que I2C en temas de transmisión además de ser fácil de implementar desde el punto de vista hardware. El protocolo I2C trabaja con un bus síncrono de datos, con un sistema de tipo maestro-esclavo y es bastante utilizado debido a la gran cantidad de dispositivos que tienen este tipo de interfaz. Algunas de las diferencias más significativas son la velocidad de los datos ya que SPI opera entre 10 Mbps y 20 Mbps mientras que I2C lo hace hasta 3,4 Mbps. Además, I2C requiere mucha más energía que SPI, pero es mucho más sencillo. El protocolo I2C tiene menos pines que el protocolo SPI [8].

Pycom es una compañía de desarrollo software y hardware dedicada al diseño, desarrollo y producción de dispositivos IoT y plataformas para la gestión de los mismos. Disponen de un gran número de placas que se pueden combinar entre si según la necesidad del usuario. Estas placas se pueden dividir en placas SoC y placas de aplicación con sensores o conectores de expansión. Las primeras pueden ser programadas para realizar acciones y ejecutar programas definidos en su memoria mientras que las placas de

aplicación proporcionan a los SoC acceso a periféricos externos como sensores o actuadores.

En este trabajo se han empleado la placa SoC WiPy versión 2.0 y la placa de aplicación Pysense de Pycom, que se detallaran en profundidad en las secciones “2.2.1” y “2.2.2” respectivamente.

2.2.1 WiPy

El SoC WiPy 2.0 [9], ilustrado en la Figura 2-1, es una placa microcontroladora de 42x20x2,5 mm y un peso de 5g desarrollada por Pycom. WiPy se basa en el SoC Espressif ESP32 [10] que posee un procesador dual de 32 bits a 8 MHz, 4 MB de RAM y 8 MB de memoria flash para el programa. Incorpora conectividad vía Bluetooth versión 4.2 normal y de baja energía (low energy, BLE) y WiFi 802.11b/g/n hasta 16 Mbps con un alcance de hasta 1 Km. El microcontrolador se programa empleando el lenguaje de programación MicroPython.

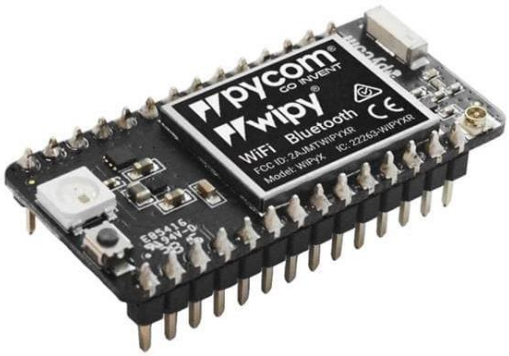

Figura 2-1: WiPy de Pycom [9]

En cuanto a periféricos, dispone de 24 pines Cuenta con 4 temporizadores de 16 bits con PWM (Pulse Width Modulation) y con 8 canales analógicos ADC (Analogic Digital Conversor) de 12bits cada uno .También dispone de I2C, SPI y I2. Estos últimos son protocolos de comunicación que establecen como deben de ser las interconexiones de dispositivos, por eso también se habla de interfaz I2C o SPI, por ejemplo.

Soporta diversos estándares de cifrados como SHA, MD5, DES o AES que permiten proteger las comunicaciones entre el dispositivo y la aplicación IoT.

Este SoC puede ser conectado a placas de aplicación como por ejemplo “Expansion Board 2.0” o la placa “Pysense” que se describe en la sección 2.2.2.

2.2.2 Pysense

La placa de aplicación Pysense [11], ilustrada en la ***Figura 2-2***, está diseñada para poder conectarse a las placas microcontroladoras IoT fabricadas por Pycom como es el caso de la placa WiPy, con el fin de proporcionarle acceso a sensores de medidas ambientales. Se trata de una placa de pequeño tamaño (55x35x10 mm) y un peso de 11 gramos que cuenta con un conector y circuito de carga para una batería externa, un puerto USB que emula un puerto serie virtual e incorpora una ranura para conectar una tarjeta microSD donde la placa microcontroladora puede almacenar datos, pero no aplicaciones. Además, tiene la capacidad de operar en modo bajo consumo de energía para ahorrar batería.

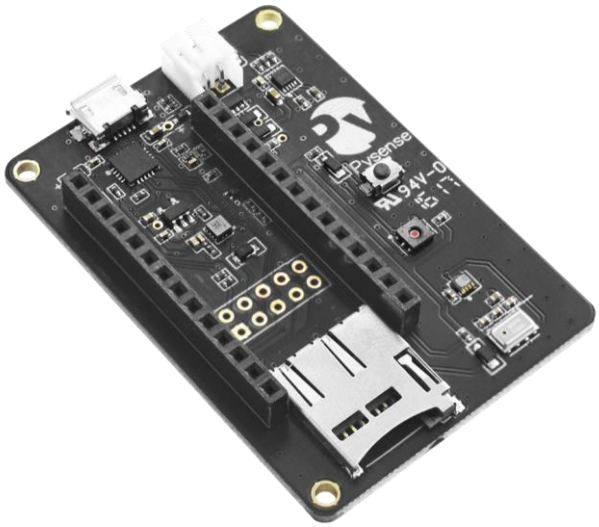

Figura 2-2: Pysense de Pycom [11]

La placa de aplicación Pysense incorpora cuatro sensores: *i) el sensor de luz ambiental LTR-329ALS-01, ii) el sensor acelerómetro LIS2HH12, iii) el sensor de humedad y temperatura SI7006A20, y iv) el sensor de presión y altitud MPL3115A2*. Estos sensores, se describen a continuación.

i) El sensor de luz ambiental LTR-329ALS-01 [12] que tiene un rango dinámico de medida entre 0.01 lux y 64k lux con una resolución efectiva de 16 bits. Se conecta al SoC por medio de un bus I2C. Opera a un voltaje de entre 2.4V y 3.6V y soporta un rango de temperaturas de operación entre -30ºC y 70ºC.

ii) El acelerómetro LIS2HH12 [13] es un sensor de montaje superficial (SMD) que dispone de un ancho de banda de 5-400 Hz y una temperatura de operación de -40ºC a 85ºC. Dispone de salida I2C y SPI. El acelerómetro consta de 3 ejes con una resolución de 11 bits. El voltaje de la fuente puede ser de 1.71V a 3.6V.

iii) El sensor de humedad y temperatura, SI7006A20 [14], es un chip de montaje superficial que está dotado de una interfaz I2C. Tiene 14 bits de resolución en la medida. Puede operar con voltajes de 1.9V a 3.6V y temperaturas entre -40ºC y 125ºC. Consume poca potencia: 150 uA cuando la corriente está activa y 60 nA cuando se encuentra en Stand-By.

iv) El sensor de presión y altitud MPL3115A2 [15] tiene una resolución para cada una de las medidas es de 20 bits. La salida se muestra en una interfaz I2C y en formato de 24 bits. Opera con temperaturas entre -40ºC a 85ºC y voltajes de 1.95V a 3.6V.

2.3 MQTT y Eclipse Mosquitto

El protocolo de encolado de mensajes para transporte de telemetría (Message Queuing Telemetry Transport, MQTT) es un protocolo de comunicación diseñado para ser muy ligero y poder ser implementado en pequeños dispositivos con recursos muy limitados. De esta forma, consume un ancho de banda muy pequeño y muy poca energía del microcontrolador. Este protocolo es de acceso libre y es perfecto para aplicaciones IoT, ya que lo puede implementar casi cualquier tipo de dispositivo.

El funcionamiento del protocolo MQTT se basa un servidor MQTT o "broker" que hace de gestor de mensajería, y una serie de clientes. El servidor enruta los mensajes que recibe

de unos clientes, los publicadores, a otros que se hayan subscrito a dicho tipo de mensajes, los subscriptores. Para ello, los clientes que mandan mensajes asignan un tópico al mensaje, que el servidor emplea para asignar el mensaje a una cola concreta, donde este se almacena de forma temporal, y reenviarlo a los clientes subscriptores de dicha cola. Como ya hemos mencionado, los clientes pueden operar como publicadores cuando encolan mensajes en tópicos concretos, o como subscriptores, si se subscriben a tópicos concretos y se quedan esperando a recibir mensajes. Cuando un publicador envía un mensaje al bróker para un tópico concreto, dicho mensaje será retransmitido a todos los subscriptores que atiendan a ese tópico. Un cliente concreto puede operar a la vez como subscriptor y como publicador si fuera necesario.

Mosquitto es un bróker de mensajería con soporte para MQTT comúnmente empleado en sistemas basados en IoT. Es un software de código abierto, por lo que se utilizará, junto con MQTT, en el desarrollo de este trabajo.

Este protocolo tiene diferentes librerías dependiendo del entorno sobre el que se va a usar. Paho [16] es una librería de código abierto para MQTT mantenida por Eclipse que permite el uso de este protocolo de mensajería en multitud de lenguajes de programación, entre ellos Python, que es la que hemos usado para este trabajo.

Por otro lado, se ha usado otra librería alternativa para la placa de Pycom. MQTTClient [17] es una versión de este que nos permitirá usar Mosquitto desde nuestro dispositivo Pycom. Se ha usado debido a que es la que proporciona y recomienda Pycom para sus placas y Paho no está disponible para MicroPython.

2.3.1 Comparativa con HTTP

El Protocolo de Transferencia de Hipertexto (Hypertext Transfer Protocol, HTTP) [18] es un protocolo que permite intercambiar información entre un cliente y un servidor. El funcionamiento del protocolo está basado en operaciones de solicitud y respuesta, de tal forma que el cliente y el servidor establecen una conexión mediante varios mensajes de solicitud, enviados por el cliente, que son respondidos por el servidor. Dado que es un protocolo ampliamente utilizado también para la comunicación entre sistemas informáticos, se ha realizado una comparación para valorar las ventajas de MQTT sobre HTTP [19].

En la ***Tabla 2-1*** podemos ver una comparativa con las características más importantes de ambos estándares:

Tabla 2-1: Comparación entre HTTP y MQTT

Característica	HTTP	MQTT
Estilo	Pregunta / Respuesta	Publicación / Subscripción
Verbos	GET / POST / DELETE	Pub / Sub / Unsub
Tamaño de Mensaje	Mensajes largos, muchos datos en cabeceras	Mínimo de 2 bytes en las cabeceras
Calidad de Servicio	Ninguna (codificación específica en la aplicación)	3 niveles
Distribución de Datos	1-a-1	1-a-1, 1-a-ninguno, 1-a-n

MQTT es un protocolo que está centrado en los datos y mensajes, mientras que el protocolo HTTP se centra más en los documentos y/o recursos.

Como podemos ver en la ***Tabla 2-1***, HTTP es un protocolo de solicitud y respuesta para la comunicación basada en el modelo de cliente-servidor, esto quiere decir que los dos extremos siempre tienen que estar operativos, cosa que no necesariamente sucederá en sistemas basados en IoT. Además, en el caso de IoT, donde puede ser necesario comunicar con cientos o miles de dispositivos, una aplicación basada en HTTP debería mantener una cantidad proporcional de conexiones activas. Por todo lo anterior, se ve claramente que no está pensado para dispositivos IoT por lo que no siempre será la mejor opción. MQTT, por el contrario, transfiere datos como una cadena de bytes en binario consiguiendo una alta ligereza de transmisión; además, el modelo de publicación-subscripción permite que la aplicación y dispositivos no tengan que estar activos todo el tiempo, y permite que la aplicación pueda atender los mensajes recibidos según vaya teniendo recursos de cálculo disponibles reduciendo las probabilidades de quedar desbordada de peticiones. Además, permite que la aplicación solo tenga que comunicar con el bróker de mensajería y que sea este último el que se encargue de servir las peticiones a los dispositivos cuando estos se activen. Por todo ello, es una opción muy conveniente para dispositivos con recursos limitados debido a que el dispositivo utilizará los tópicos cuando los necesite sin necesidad de una conexión continua y por tanto, ahorrando batería.

Como se puede ver, el modelo de publicación-subscripción proporciona a los dispositivos que no garantizan una conexión continua a la red una forma de poder escuchar y enviar mensajes. No solo eso, además hace que los clientes tengan una existencia independiente entre sí ya que no es necesaria una comunicación directa entre ambos. Esto permite que, aun cuando un cliente está fuera de servicio, todo el sistema puede seguir funcionando correctamente.

La calidad de servicio es algo bastante importante en los protocolos de mensajería. Mientras que en el protocolo HTTP no se gestiona ninguna calidad de servicio, ya que depende de la codificación específica de la aplicación, en MQTT podemos encontrar tres niveles de servicio que nos proporciona dicho protocolo:

- **A lo sumo una vez:** Este método garantiza que el mensaje solo se va a entregar una vez, esto quiere decir que dicho mensaje no se confirma. Es el método más rápido, pero el menos fiable.

- **Al menos una vez:** El mensaje será entregado al menos una vez, pero puede ser entregado más de una vez. El emisor manda el mensaje y espera a una confirmación de que ha sido recibido, si la recibe lo notifica y si no lo recibe vuelve a mandar el mensaje hasta que reciba respuesta.

- **Exactamente una vez:** Garantiza que cada mensaje sea recibido solo una vez por la otra parte, pero es el más lento ya que requiere más mensajes. El emisor envía el mensaje y está a la espera de la confirmación del mismo. El receptor contesta al emisor cuando recibe el mensaje con un mensaje de confirmación que el emisor debe liberar para que el receptor sepa que todo ha ido correctamente.

Mientras que el protocolo HTTP no muestra diferentes habilidades u opciones en caso de que ocurra algo inesperado o predefinido, MQTT ofrece varias opciones de mensajes. Por ejemplo, cuando ocurre una desconexión inesperada de un cliente, la función de "última voluntad" enviará a los clientes subscritos un mensaje determinado; otro ejemplo, es la función de "retenido", que hace que un cliente recién subscrito reciba una actualización de estado inmediata, de forma que el último mensaje enviado (retenido) se va actualizando para que los nuevos clientes lo reciban.

En cuanto al tamaño de mensajes, MQTT tiene una especificación bastante corta ya que solo existen cinco tipos de mensajes: conexión, publicación, subscripción, des-subscripción

o desconexión. Por el contrario, las especificaciones HTTP tienen muchas más acciones y tipos de mensaje como GET, PUT, POST, etc. MQTT tiene un encabezado de mensaje muy corto y el tamaño de mensaje de paquete más pequeño es de 2 bytes.

En conclusión, el protocolo MQTT parece mejor elección que HTTP cuando se necesita que el tiempo de respuesta, rendimiento, ahorro de batería y ancho de banda sean lo más óptimos posible, como sucede con los dispositivos IoT.

2.4 InfluxDB

InfluxDB [4] es una base de datos de código abierto destinada a almacenar datos de series temporales desarrollada por la compañía InfluxData. InfluxDB es capaz de manejar grandes cantidades de información a gran velocidad y está especialmente diseñada para sistemas de monitorización e IoT.

Típicamente, las bases de datos tratan con objetos relacionales, es decir, que están vinculados entre ellos por medio de claves foráneas. Para manejar y gestionar los sistemas de bases de datos relacionales, es común emplear lenguajes de consulta de bases de datos como el Lenguaje de Consulta Estructurada (Structured Query Language, SQL). Este tipo de lenguajes permiten al programador definir, leer o manejar datos para estructurar tablas cuyas entradas guardan relación entre sí. Aunque los datos que se almacenan en InfluxDB no son relacionales, InfluxDB proporciona un lenguaje basado en SQL para realizar dichas consultas de una forma similar a las bases de datos relacionales para facilitar el trabajo a programadores que ya estén familiarizados con SQL.

Todas las medidas que contiene una base de datos de InfluxDB siempre tienen una etiqueta temporal, una serie de etiquetas configurables por el usuario que permiten realizar filtrados de los datos, por ejemplo, el identificador del dispositivo que tomó cierta medida, o la localización del mismo, y una serie de valores (normalmente numéricos) que contienen los valores de las medidas.

Esta herramienta ha sido escogida para este trabajo ya que nos da la posibilidad de almacenar grandes cantidades de información recogida por parte de los sensores durante un largo periodo de tiempo de forma muy fácil y se integra perfectamente con Grafana, explicado en el anexo B.3 Conectar y configurar InfluxDB.

2.5 Grafana

Grafana [5] es una plataforma de código abierto que permite la creación de cuadros de mando usando distintos tipos de gráficos para visualizar los datos almacenados en bases de datos de distintos tipos. Soporta bases de datos de series temporales, como es el caso de InfluxDB, explicado en la sección 2.4. También ofrece la posibilidad de establecer alertas y alarmas personalizadas para cada uno de los gráficos diseñados.

Grafana, al trabajar principalmente con series temporales de datos, permite realizar filtrados por rangos temporales configurables por el usuario, con períodos de agregación y refresco de datos también configurable por el usuario. Además, la herramienta también permite incorporar filtros personalizados en los cuadros de mandos en función de valores almacenados en la base de datos y aplicarlos sobre los datos que se vayan a utilizar para generar las gráficas. Esta característica es muy interesante para las aplicaciones IoT, entre otras, ya que nos permite realizar filtrados de datos según convenga. Por ejemplo, en el caso de una aplicación IoT, si se almacena el identificador del dispositivo que ha tomado las medidas y la región donde está desplegado, el cuadro de mandos puede ofrecer un par de campos de selección (por dispositivo y por región) de forma que el gráfico generado solo incluirá los valores que cumplan ambos filtros. Además, estos campos de filtrado

soportan también multi-selección, de forma que permitan seleccionar subconjuntos de entidades, es decir, en el ejemplo anterior, permitiría seleccionar un subconjunto de dispositivos o un conjunto de regiones.

Grafana dispone de variedad de gráficos definidos por defecto como gráficos de serie temporal, pastel, barras, tablas, etc. Pero también existen módulos capaces de extender la funcionalidad de Grafana añadiendo nuevos tipos de gráficos y nuevas fuentes de datos.

Por todo ello, se considera que Grafana es una buena elección para generar cuadros de mandos para este trabajo.

2.6 Docker

Docker [3] es una plataforma, de código abierto, basada en la tecnología de contenedores creada para facilitar el despliegue de software mediante el uso de contenedores. Un contenedor se traduce como un software cuya finalidad es empaquetar una aplicación software junto con sus librerías y dependencias de tal manera que la aplicación pueda ser desplegada en cualquier sistema con tecnología Docker y ser ejecutado de forma rápida y fiable.

En Docker, las aplicaciones, una vez desarrolladas, se empaquetan en imágenes para su distribución. Dichas imágenes contienen la propia aplicación, así como las librerías y ficheros de dependencias que esta requiera. En el momento de desplegar la aplicación, la imagen se desempaqueta en el sistema huésped junto con sus dependencias y se ejecuta en un entorno aislado del resto de contenedores para que no interfieran entre ellos, pero con la posibilidad de que se comuniquen a través de redes virtuales entre ellos.

El motor de Docker (Docker Engine) es el encargado de transformar estas imágenes en contenedores del propio Docker. La principal ventaja que tiene este motor de contenedores es que es independiente de la estructura del sistema donde trabaja, es decir, podrá ejecutarse tanto en un sistema operativo Linux, como en Windows, etc. Esto permite que los contenedores se aíslen del resto del equipo y trabajen de una forma independiente sin que le afecten en su funcionamiento factores externos.

Docker ofrece grandes ventajas a la hora de ejecutar un contenedor con Docker Engine en comparación con las máquinas virtuales clásicas. En primer lugar, la compañía Docker fue la creadora de todo el sistema de contenedores y el estándar de estos mismos, haciendo que sus contenedores puedan ser instalados en cualquier lugar. Los contenedores han sido diseñados de tal manera que son independientes de la máquina donde se ejecutan pero pueden configurarse para compartir el núcleo del sistema huésped, esto quiere decir que no necesitan desplegar otro sistema operativo, como pasa con las máquinas virtuales, haciendo que la eficiencia del sistema huésped aumente notablemente.

2.7 Conclusiones

En esta sección se han presentado las tecnologías que se van a utilizar a lo largo del desarrollo del sistema.

En primer lugar, se ha presentado el concepto de Internet de las Cosas y de los dispositivos que se han utilizado a lo largo del trabajo, como el SoC WiPy 2.0 y la placa de aplicación Pysense. También se ha presentado el protocolo que se va a utilizar en el desarrollo del sistema, MQTT: así como el bróker de mensajería que se ha aplicado, Mosquitto. La comparativa realizada entre MQTT y HTTP ha ayudado a verificar que el protocolo escogido sea óptimo para el uso de dispositivos IoT.

Finalmente, se ha hablado de diferentes herramientas para el almacenamiento y representación de datos que serán útiles en el desarrollo completo del sistema, como es el caso de InfluxDB para almacenar series temporales de muestras, Grafana para visualizarlas, y Docker para levantar los contenedores de InfluxDB y Grafana.

3 Diseño y Desarrollo

En este capítulo se presentará, en primer lugar, los detalles técnicos del sistema IoT para la adquisición de medidas ambientales que se ha diseñado y desarrollado en este trabajo final de grado. Se describirá la arquitectura del sistema, de la aplicación IoT desarrollada y del aplicativo de control del dispositivo IoT empleado. Además, se detallará la especificación del protocolo diseñado y desarrollado para llevar a cabo la comunicación entre la aplicación y el dispositivo IoT.

3.1 Arquitectura del Sistema

La arquitectura del sistema IoT de adquisición de medidas ambientales desarrollado en este trabajo se ilustra en la Figura 3-1, y consta de tres componentes principales: la aplicación IoT, el gestor de mensajería y los dispositivos IoT, que se explicarán a continuación.

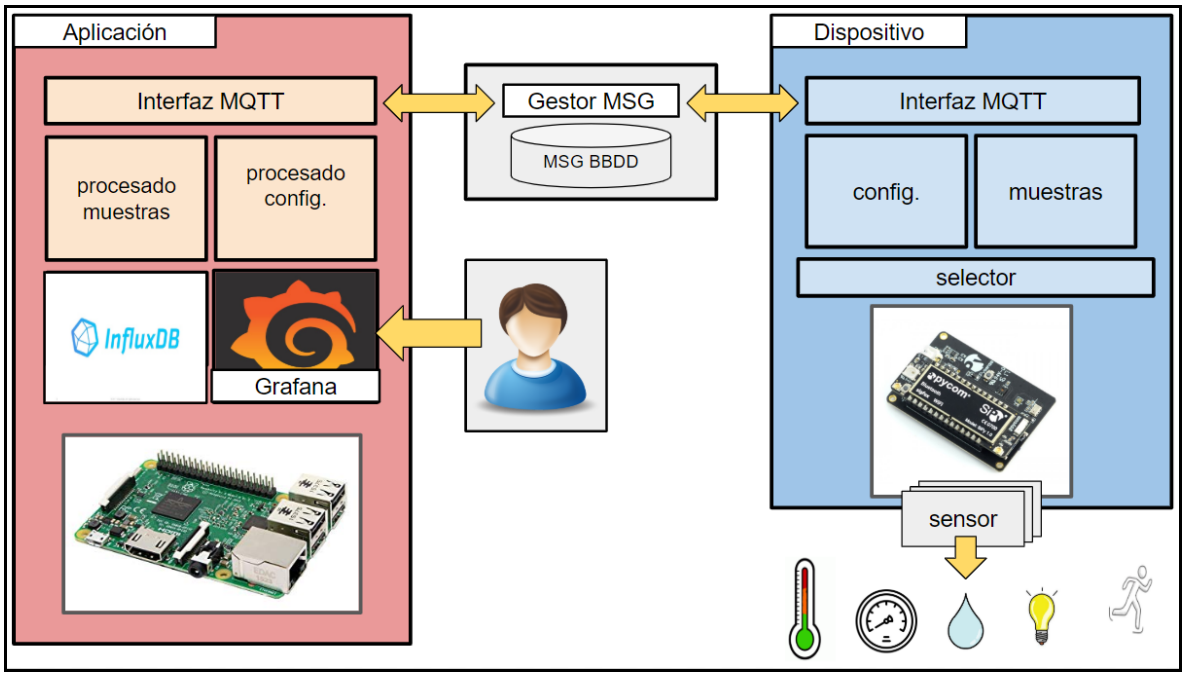

Figura 3-1: Diagrama de bloques del sistema desarrollado

La aplicación consiste en una “Raspberry 3b” y el dispositivo IoT está formado por el SoC WiPy y una placa de aplicación Pysense de la compañía Pycom. La aplicación que corre en la Raspberry podrá comunicarse con la WiPy a través del bróker de mensajería de Mosquitto. Finalmente, la WiPy podrá tomar medidas ambientales gracias a la placa de aplicación que contiene a los sensores de temperatura, presión y altitud, humedad, luz ambiente y al acelerómetro equipados en la Pysense donde la WiPy está conectada.

En primer lugar, se ha establecido una conexión física a través de un cable USB entre la Raspberry y la Pycom para poder alimentar a esta última. Esta conexión no es relevante para el intercambio de mensajes entre ambas placas. Además, se van a utilizar los protocolos SFTP y SSH.

La primera conexión entre el usuario y el módulo IoT se establece mediante el protocolo SSH que permite manejar la Raspberry y trabajar con ella de forma remota. Para acceder desde esta hasta la placa de Pycom, se han utilizado diferentes ejecutables de

BASH. Con ellos se puede subir el código a la Pysense, ejecutar instrucciones desde esta o usar un telnet. El protocolo SFTP se ha utilizado para subir códigos o ejecutar procesos.

La aplicación IoT es la encargada de controlar el sistema completo, almacenar los datos, presentar los datos al usuario, e interactuar con los dispositivos IoT para controlarlos. Esta se compone de los siguientes componentes:

i) La *Interfaz MQTT* es la encargada de intercambiar los mensajes en los tópicos que correspondan con el bróker para poder interactuar con los dispositivos IoT.

ii) El módulo *Procesado de Muestras* es el encargado de las operaciones con las muestras, es decir, solicitar las muestras necesarias a los dispositivos y almacenar las muestras recibidas.

iii) El módulo *Procesado de Configuración* es el encargado de enviar actualizaciones de la configuración de los dispositivos o recibir los parámetros de configuración de estos cuando la aplicación lo requiera.

iv) *InfluxDB* es la base de datos de series temporales donde se almacenan los datos recibidos de los sensores y empleados por Grafana para componer y presentar al usuario los gráficos con los datos. En el apéndice B.3 Conectar y configurar InfluxDB se describe como conectar con InfluxDB y como se ha empleado para almacenar cada una de las muestras.

v) *Grafana* es la aplicación empleada para realizar la visualización de datos almacenados en *InfluxDB*. En el apéndice B.4 Configurar gráficos en Grafana se explica cómo añadir los gráficos en el cuadro de mandos de Grafana para visualizar los datos.

Por otro lado, el dispositivo, en nuestro caso está formado por las placas de Pycom WiPy y Pysense. Cuando el dispositivo recibe un mensaje de solicitud de muestras , tiene que obtener las medidas de los sensores correspondientes. Estos sensores se encuentran en una placa de aplicación de Pycom, la Pysense, que está unida a la placa microcontroladora WiPy. La conexión entre ambas placas es física por lo que los datos provenientes de los sensores serán traspasados de la placa de aplicación a la placa microcontroladora mediante un bus de datos en una interfaz I2C o SPI dependiendo del tipo de protocolo de comunicación del sensor. El dispositivo está compuesto por los siguientes componentes:

i) La *Interfaz MQTT* es la encargada del intercambio de mensajes con el gestor de mensajería.

ii) El módulo *Configuración* es el encargado de suministrar información sobre la configuración actual de la placa y de cambiar dicha configuración si se recibe alguna petición para ello.

iii) El módulo *Muestras* se encarga de recibir los mensajes de petición de muestras y activar los sensores para que recojan la medida solicitada; para ello debe identificar el tipo de muestra solicitada y seleccionar el sensor que debe tomar la medida, así como sus características y dirección, con la finalidad de pasar esta información al "Selector".

iv) El módulo *Sensores* es el que contiene a todos los sensores del dispositivo y del cual se obtienen todas las medidas ambientales.

Este selector es un método para la identificación de sensores. Para ello, el software ha sido diseñado para que separe el identificador de cada elemento que reciba. Dependiendo del valor de sus dos bytes más significativos podrá saber si se trata de la dirección de un sensor o de un componente de la placa. Si los dos bytes seleccionados tienen un valor de

"0x01" se trata de la dirección de un señor, por lo que se continuará leyendo el resto de bits para saber de qué sensor se trata.

Tras lanzar la aplicación y activar el dispositivo, ambos establecen una conexión mediante el protocolo MQTT con Mosquitto. Este bróker de Mosquitto es el módulo Gestor MSG, el cual hace posible toda la comunicación entre los dos extremos. Para ello dispone de una pequeña base de datos con los mensajes que pasan por él. Solo interfiere con las interfaces MQTT de la aplicación y el dispositivo. Como gestor de mensajería se ha empleado Eclipse Mosquitto; éste emplea el protocolo MQTT para permitir que sus publicadores y subscriptores puedan intercambiar mensajes. En particular, en este sistema, tanto la aplicación IoT como los dispositivos IoT son tanto publicadores como subscriptores. En la sección 3.3 se presenta el protocolo desarrollado junto con un diagrama de flujo explicativo donde se ilustran las interacciones entre la aplicación y los dispositivos, así como los tópicos empleados en estas interacciones.

3.2 Implementación del Código

Los lenguajes de programación empleados en todo el trabajo han sido Python [20] y MicroPython, ya que es un lenguaje portable entre sistemas operativos que contiene multitud de librerías útiles ya preparadas para que el usuario la utilice.

- **Application.py :** Implementa la funcionalidad de la aplicación. Cuando un dispositivo se activa, manda un mensaje que recibe este módulo y desencadena la solicitud de muestras al dispositivo [21].
- **ApplicationDefinitions.py :** Implementa diversos métodos y funciones necesarios para enviar peticiones de muestras y/o de configuraciones. Además, también se encarga de recibir y almacenar las muestras recibidas de los dispositivos.
- **Device.py :** Implementa la conexión del dispositivo con el gestor de mensajería y la función de callbak para recibir y clasificar los mensajes del gestor. Además, realiza algunas subscripciones previas.
- **DeviceDefinitions.py :** Contiene métodos para enviar los sensores activos en el dispositivo. También se encarga de realizar las medidas de los sensores y de empaquetar los mensajes con las muestras tomadas [22].
- **Common.py :** Este archivo contiene variables, parámetros y constantes que son comunes al dispositivo y a la aplicación. Entre otros, define los tópicos que se van a utilizar en los mensajes, así como las direcciones de los identificadores de los sensores conocidos por la aplicación IoT.

3.3 Especificación del protocolo de comunicaciones.

La comunicación de la aplicación y el dispositivo con el broker de mensajería queda mostrada en el diagrama de la ***Figura 3-2*****¡Error! No se encuentra el origen de la referencia.** donde se pueden apreciar los mensajes que se intercambian para realizar dicha comunicación.

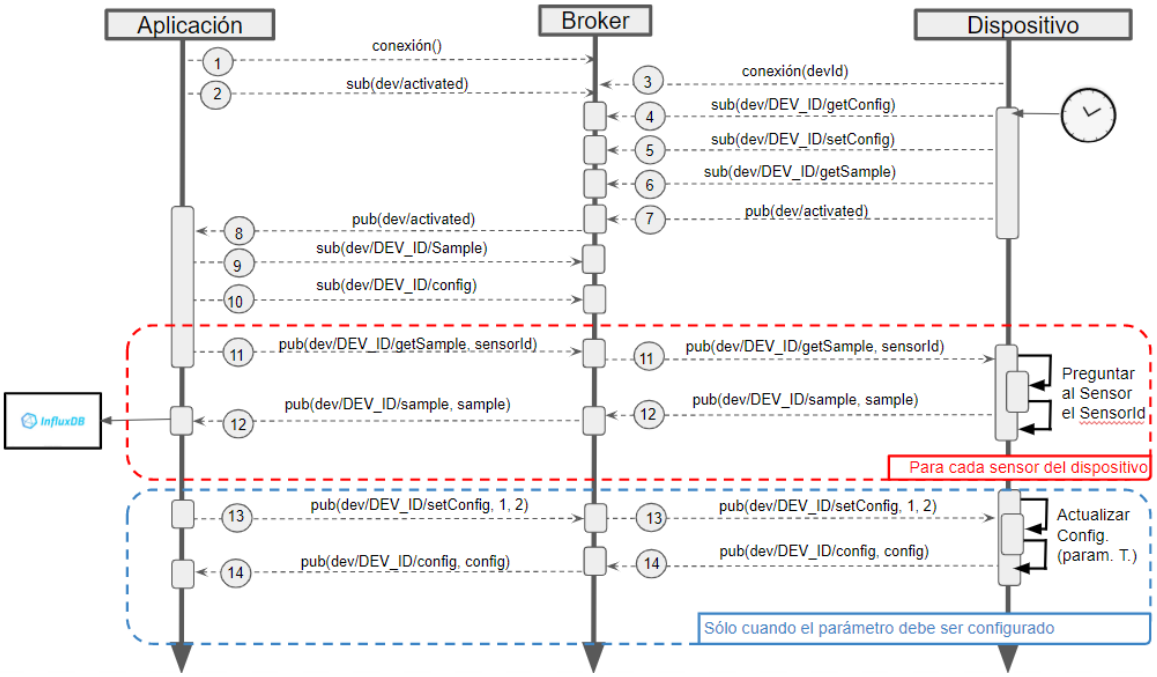

Figura 3-2: Diagrama Completo

En la ***Figura 3-2*** se ilustra el proceso de conexión de la aplicación IoT al bróker de mensajería (etiquetado como 1 en la figura), y su subscripción al tópico “dev/activated” (2). También se muestra la conexión del dispositivo IoT al bróker (3)

La conexión desde la aplicación al gestor de mensajería se realiza con la librería paho-mqtt instalada en la Raspberry.

La configuración de la comunicación de nuestro cliente y el gestor de mensajería requiere ciertos valores de dicho cliente. En nuestro caso la IP utilizada es la "172.254.1.15” que corresponde con la dirección IP Wifi de la Raspberry y el puerto establecido el “1883”. Además, se establecerá un usuario y contraseña para poder utilizar restricciones de seguridad que nos ofrece el broker.

Por otro lado, la conexión del dispositivo al agente de mensajería es bastante similar al anterior ya que se trata del mismo protocolo pero con algunas variaciones debido a la librería utilizada. La definición del cliente utiliza los mismos parámetros que el caso anterior salvo por la utilización del identificador del dispositivo en formato cadena. Una vez definido el cliente se realiza la conexión.

Cuando el dispositivo se despierta, se subscribe a diferentes categorías de mensajes o tópicos para poder recibir mensajes de la aplicación al otro lado del broker de mensajería: “getConfig” (etiquetado como 4 en la ***Figura 3-2*** para poder obtener mensajes de petición de configuración, “setConfig” (5) para poder obtener mensajes solicitando modificar la configuración del dispositivo y “getSample” (6) para poder recibir peticiones de mediciones solicitadas por la aplicación.

Además de estas subscripciones, el dispositivo publica un mensaje en el tópico *“dev/activated/* indicando que se encuentra activo. El mensaje publicado contiene el identificador único de dispositivo y una lista con los identificadores de los sensores disponibles en el dispositivo, mensaje que será entregado íntegramente a cualquiera que se haya subscrito al tópico del Broker (7). El formato de mensaje se detalla en la sección 3.3.1.1.

Una vez el dispositivo ha notificado su activación y ha facilitado a la aplicación su identificador y la lista de sensores que tiene equipados, la aplicación se subscribe a los tópicos vinculados con el dispositivo gracias al identificador proporcionado por el dispositivo. Para ello, se componen los nombres de los tópicos como *"dev/DEV_ID/Sample"* que se usa para poder obtener las muestras publicadas (9) y *"dev/DEV_ID/config"* para obtener los mensajes de configuración enviados por el dispositivo (10).

Una vez la aplicación y el dispositivo se encuentran subscritos a los tópicos empleados para la comunicación, la aplicación puede ejecutar uno de los siguientes bloques que se pueden apreciar en la ***Figura 3-2***: *bloque de muestras* o *bloque de configuración*. El primer bloque se ejecutará cuando la aplicación solicite muestras obtenidas por los sensores del dispositivo, mientras que el segundo bloque se llevará a cabo cuando ocurra algún hito configurado por el usuario o simplemente porque dicho usuario quiera realizar una reconfiguración del dispositivo.

En el bloque de solicitud de muestras, la aplicación solicita los datos publicando un mensaje de solicitud de muestras indicando el identificador del sensor en el tópico (*dev/DEV_ID/getSample*) donde está subscrito el dispositivo.

Este bloque de solicitud de muestras se ejecutará para cada tipo de muestra que el dispositivo ha informado que puede proporcionar. Entonces, al recibir la notificación de activación, la aplicación publicará un mensaje de solicitud en el tópico *"dev/DEV_ID/getSample"* por cada tipo de muestra que este último ha informado (11). El formato de este mensaje queda explicado en la sección 3.3.1.2

Cuando el dispositivo recibe la solicitud de una muestra concreta, debe identificar el sensor que debe emplear para la toma de la medida, tomar propiamente la medida, y, posteriormente, crear un mensaje que será publicado en el tópico *"dev/DEV_ID/sample"* para que la aplicación la reciba (12). Dependiendo del tipo de muestra tendremos formatos diferentes de mensajes. Los formatos implementados para este sistema, y las secciones donde se detallan, son los siguientes: *i*) luz ambiental (sección 3.3.1.3), *ii*) acelerómetro (sección3.3.1.4), *iii)* humedad y temperatura (sección 3.3.1.5), y *iv*) presión y altitud (sección 3.3.1.6*)*

Por otro lado, el proceso de configuración sólo se llevará a cabo si hay algún parámetro que la aplicación necesite reconfigurar en el dispositivo. La aplicación pedirá una nueva configuración a la placa mediante la publicación de un mensaje en el tópico "dev/DEV_ID/setConfig"(13).

Una vez el dispositivo ha recibido la nueva configuración y la ha establecido, publica un mensaje a modo de comprobación informando a los que escuchen en el tópico "dev/DEV_ID/config" que la reconfiguración se ha realizado correctamente indicando el nuevo valor del parámetro cambiado (14).

En la ***Tabla 3-1*** se resumen todos los tópicos empleados junto con una pequeña descripción de los mismos, así como el subscriptor y publicador relacionados.

Tabla 3-1: Tópicos Utilizados

Tópico	Publicador	Subscriptor	Descripción
dev/DEV_ID/getConfig	Aplicación	Dispositivo	Solicitud de configuración actual.
dev/DEV_ID/setConfig	Aplicación	Dispositivo	Solicitud de reconfiguración.
dev/DEV_ID/getSample	Aplicación	Dispositivo	Solicitud de medida.
dev/activated	Dispositivo	-	Notificación de activación de dispositivo.
dev/DEV_ID/sample	Dispositivo	Aplicación	Respuesta a solicitud de medida.
dev/DEV_ID/config	Dispositivo	Aplicación	Respuesta a solicitud o cambio de configuración.

3.3.1 Formato de los mensajes

En esta sección se detalla el formato de cada tipo de mensaje definido en el protocolo detallando a nivel de bits los campos que los componen, así como su contenido. Es importante notar que todos los campos se codifican a nivel de bits y que los bytes de cada campo están ordenados en orden natural de red (conocido en inglés como ordenación big-endian) [23]

3.3.1.1 Mensaje de activación

El mensaje mostrado en la ***Figura 3-3*** , hace referencia a la primera publicación que se lleva a cabo cuando se activa un dispositivo (etiquetado como 4 en la ***Figura 3-3***). Este mensaje tiene una longitud variable debido a dependerá del número de sensores que estén equipados en el dispositivo.

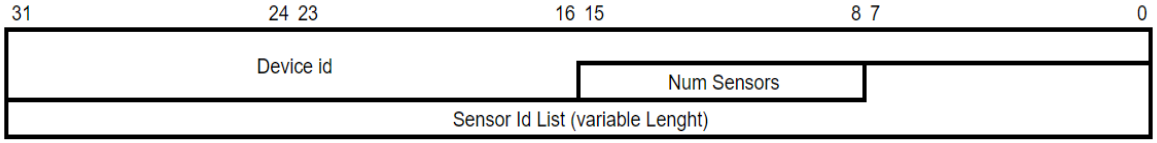

Figura 3-3: Mensaje de Activación

El mensaje está formado por los siguientes campos:

- **Device Id (6 Bytes):** contiene el identificador del dispositivo. Empleamos la dirección física de control de acceso al medio (Medium Access Control, MAC) del dispositivo que envía el mensaje codificada a nivel de bits.
- **Num Sensors (1 Byte):** indica el número de sensores equipados en el dispositivo y, por tanto, se corresponde con el tamaño de la lista de sensores que nos mostrará el siguiente campo.
- **Sensor Id List (longitud variable):** consiste en una lista de enteros cortos (2 bytes cada uno) que representan los sensores equipados que tiene el dispositivo. Debido a que cada dispositivo es diferente el tamaño de este campo es variable.

3.3.1.2 Mensaje de solicitud de muestra

El mensaje que publica la aplicación para solicitar muestras al dispositivo (7), se ilustra en la ***Figura 3-4***. Su tamaño es de 2 bytes..

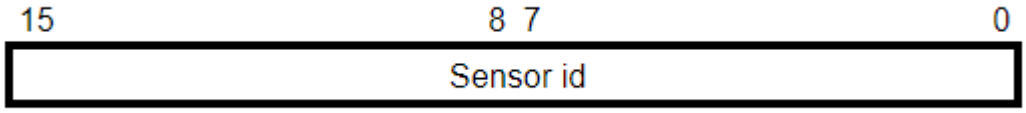

Figura 3-4: Mensaje de Solicitud de Muestras

El mensaje está formado por el siguiente campo:

- **Sensor Id (2 Bytes):** contiene la dirección del dispositivo debe reportar la medida.

3.3.1.3 Respuesta de muestra del sensor de luz ambiental

La ***Figura 3-5*** muestra el mensaje que publica el dispositivo en respuesta a una solicitud de medida del sensor de luz ambiental. En él se encapsulan 6 bytes que nos mostrarán las mediciones tomadas por el sensor.

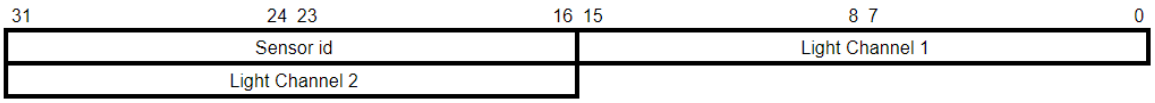

Figura 3-5: Mensaje del Sensor de Luz Ambiental

El mensaje está formado por los siguientes campos:

- **Sensor Id (2 Bytes):** contiene la dirección del sensor de luz ambiental.
- **Light Channel 1 (2 Bytes):** contiene un entero corto con la medida sobre el canal rojo (longitudes de onda de entorno a 770 nm) del sensor.
- **Light Channel 2 (2 Bytes):** contiene un entero corto con la medida sobre el canal azul (longitudes de onda entorno a 450 nm).

3.3.1.4 Respuesta de muestra del sensor acelerómetro

Las medidas recogidas por el acelerómetro son enviadas hacia la aplicación mediante un mensaje de 22 bytes como el anterior, ***Figura 3-6***.

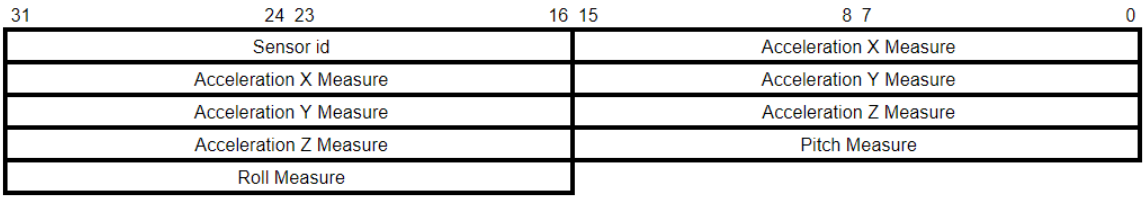

Figura 3-6: Mensaje del Acelerómetro

El mensaje está formado por los siguientes campos:

- **Sensor Id (2 Bytes):** contiene el identificador del sensor
- **Acceleration X Measure (4 Bytes):** contiene la medida de aceleración en el eje X.
- **Acceleration Y Measure (4 Bytes):** contiene la medida de aceleración en el eje Y.
- **Acceleration Z Measure (4 Bytes):** contiene la medida de aceleración en el eje Z.
- **Roll Measure (4 Bytes):** contiene la medida de balanceo del acelerómetro.
- **Pitch Measure (4 Bytes):** contiene la medida de inclinación del acelerómetro.

3.3.1.5 Respuesta de muestra del sensor de humedad y temperatura

La ***Figura 3-7*** muestra el mensaje que publica el dispositivo cuando la medida solicitada se trata de la humedad del ambiente o la temperatura de este mismo, el dispositivo tomará estas medidas y las encapsulará en un mensaje como el anterior para enviarlo al gestor. Este mensaje tiene un tamaño de 10 Bytes distribuidos de la siguietne manera:

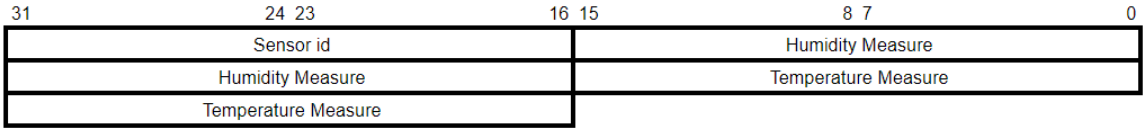

Figura 3-7: Mensaje del Sensor de Humedad y Temperatura

El mensaje está formado por el siguiente campo:

- **Sensor Id (2 Bytes)** : Dirección del sensor del sensor de humedad y temperatura (SI7006A20)
- **Humidity Measure (4 Bytes) :** Campo que recoge las medidas de humedad del sensor de la Pysense.
- **Temperature Measure (4 Bytes) :** Medidas de tempertura proporcionadas por el SI7006A20.

3.3.1.6 Respuesta de muestra del sensor de presión y altitud

Finalmente, el mensaje de la ***Figura 3-8*** publica las medidas de presión barométrica y altitud son enviadas con el formato anterior en un mensaje de 10 Bytes.

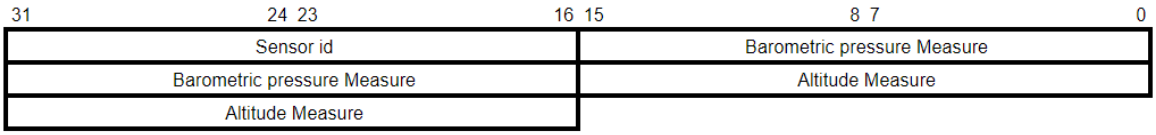

Figura 3-8: Mensaje del Sensor de Presión y Altitud

El mensaje está formado por los siguientes campos:

- **Sensor Id (2 Bytes)**: Dirección del sensor del sensor de presión barométrica y altitud (MPL3115A2)
- **Barometric Pressure Measure (4 Bytes):** Medidas de presión recogidas por el sensor.
- **Altitude Measure (4 Bytes):** Este campo contiene los datos de altitud que nos proporciona el sensor.

3.4 Conclusiones

En este capítulo se ha descrito el diseño y desarrollado de la arquitectura del sistema. En el apartado hardware se ha explicado las conexiones físicas que se han establecido y en el software todo el código necesario para la ejecución de dicho sistema.

El diagrama de conexiones se ha explicado la especificación del protocolo utilizado para el funcionamiento del sistema y ver como interactúan la aplicación y el dispositivo con el broker de mensajería.

Además, se ha especificado el formato de cada uno de los mensajes que se han publicado en los correspondientes tópicos. De tal forma, que para cada muestra se tiene en cuenta cada medida de los campos que devuelven los sensores.

4 Integración, pruebas y resultados

En este capítulo, se detallan las pruebas realizadas para validar el correcto funcionamiento del sistema. En particular, se detallarán los mensajes por pantalla que se han establecido a lo largo del programa, los resultados obtenidos en cada mensaje mostrado por consola durante la ejecución del software y, como resultado final, los datos obtenidos mostrados en una herramienta que muestra dicha información de forma gráfica.

4.1 Prueba de arranque del sistema y detección de dispositivo activado

En primer lugar se lanzará el software para poder ver por consola todos los mensajes de control establecidos y comprobar que funciona de manera correcta obteniendo los valores esperados en cada punto.

La ejecución se realiza lanzando por consola los archivos Python de cada dispositivo. Si todo ha funcionado correctamente podremos observar mensajes de la consola en las dos terminales donde se han lanzado los códigos, de tal manera que se puede apreciar por consola el valor de los campos según se van enviando y recibiendo.

Para asegurarse de que todo se ha enviado correctamente, desde ambos lados de la conexión, se ha utilizado la herramienta WireShark con el fin de realizar una captura en las interfaces de los dispositivos. En esta captura se deberían apreciar los mismos mensajes que hemos visto anteriormente enviados desde las direcciones y puertos correspondientes.

No.	Time	Source	Destination	Info
13	1.266	172.254.1.14	172.254.1.15	Connect Command
16	1.266	172.254.1.15	172.254.1.14	Connect Ack
23	1.326	172.254.1.14	172.254.1.15	Subscribe Request (id=1) [device/30:AE:A4:00:86:24/getconfig]
25	1.327	172.254.1.15	172.254.1.14	Subscribe Ack (id=1)
30	1.387	172.254.1.14	172.254.1.15	Subscribe Request (id=2) [device/30:AE:A4:00:86:24/setconfig]
32	1.387	172.254.1.15	172.254.1.14	Subscribe Ack (id=2)
37	1.446	172.254.1.14	172.254.1.15	Subscribe Request (id=3) [device/30:AE:A4:00:86:24/getsamples]
39	1.447	172.254.1.15	172.254.1.14	Subscribe Ack (id=3)
44	1.507	172.254.1.14	172.254.1.15	Publish Message [device/activated]
46	1.507	172.254.1.15	172.254.1.15	Publish Message [device/activated]
70	1.514	172.254.1.15	172.254.1.15	Subscribe Request (id=2) [device/30:AE:A4:00:86:24/sample]
71	1.514	172.254.1.15	172.254.1.15	Subscribe Ack (id=2)
72	1.514	172.254.1.15	172.254.1.15	Subscribe Request (id=3) [device/30:AE:A4:00:86:24/config], Publish Message [device/30:AE:A4:00:86:24/getsamples]
73	1.514	172.254.1.15	172.254.1.15	Subscribe Ack (id=3)
74	1.514	172.254.1.15	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/getsamples]
75	1.514	172.254.1.15	172.254.1.14	Publish Message [device/30:AE:A4:00:86:24/getsamples]
83	1.536	172.254.1.14	172.254.1.14	Publish Message [device/30:AE:A4:00:86:24/getsamples]
87	1.561	172.254.1.15	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/getsamples], Publish Message [device/30:AE:A4:00:86:24/getsamples]
92	1.587	172.254.1.14	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/sample], Publish Message [device/30:AE:A4:00:86:24/sample]
93	1.587	172.254.1.15	172.254.1.14	Publish Message [device/30:AE:A4:00:86:24/getsamples], Publish Message [device/30:AE:A4:00:86:24/getsamples]
94	1.587	172.254.1.15	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/sample]
104	1.631	172.254.1.15	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/sample]
124	2.657	172.254.1.14	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/sample]
126	2.657	172.254.1.15	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/sample]
144	3.384	172.254.1.14	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/sample]
146	3.384	172.254.1.15	172.254.1.15	Publish Message [device/30:AE:A4:00:86:24/sample]

Figura 4-1: Captura Completa de WireShark

La ***Figura 4-1***muestra toda la mensajería recogida durante la ejecución completa de los programas. Podemos apreciar en la parte inferior de la imagen que las direcciones corresponden con las de la aplicación 172.254.1.15 y el dispositivo 172.254.1.14, además de los puertos de ambos, el 63533 de la Raspberry y el 1883 por el que interactúa el gestor de mensajería.

La ***Tabla 4-1*****¡Error! No se encuentra el origen de la referencia.**, muestra una captura detallando los pasos de ejecución del programa del dipositivo. En ella podemos ver todas las interacciones a lo largo de la ejecución completa de la prueba.

Tabla 4-1: Mensajes por consola en Dispositivo

```
1  connecting to broker
2  Subscribing to topic device/30:AE:A4:00:86:24/getconfig
3  Subscribing to topic device/30:AE:A4:00:86:24/setconfig
4  Subscribing to topic device/30:AE:A4:00:86:24/getsamples
5  Publishing Activate Message: List IdSensors
6  Publishing message to topic device/30:AE:A4:00:86:24/sample
7  Publishing message to topic device/30:AE:A4:00:86:24/sample
8  Publishing message to topic device/30:AE:A4:00:86:24/sample
9  Publishing message to topic device/30:AE:A4:00:86:24/sample
```

En la **¡Error! No se encuentra el origen de la referencia.** y en la ***Tabla 4-1*** se puede observar la conexión entre el dispositivo y el gestor de mensajería. El primer mensaje nos indica que se está conectando al broker. El mensaje de tipo Ack de la ***Figura 4-1*** indica que ha recibido el mensaje, lo que quiere decir que ha establecido la conexión.

Después de la conexión, el dispositivo se subscribía a los tópicos de "getConfig", "setConfig" y "getSamples". Las tres siguientes líneas de la ***Tabla 4-1*** hacen referencia a las subscripciones que hace el dispositivo al gestor de mensajería, que corresponden con los pasos 4, 5 y 6 mostrados en la ***Figura 3-2***.

En la siguiente captura, ***Figura 4-2*****¡Error! No se encuentra el origen de la referencia.**, se muestran estos tres mensajes de subscripción, obtenidos con una herremienta de captura de tráfico, más en detalle junto con sus respectivos mensajes Ack que nos confirman que se han recibido correctamente. También se puede comprobar que las direcciones son correctas, al igual que el tópico, que contiene la dirección física del dispositivo.

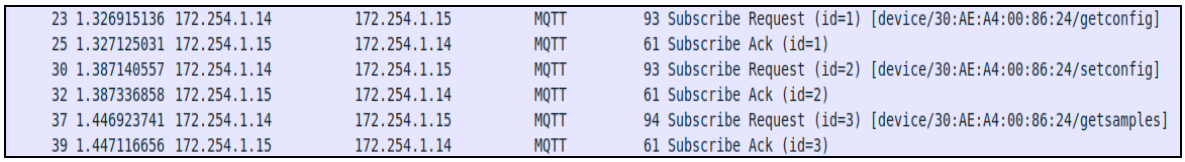

Figura 4-2: Detalle mensajes de subscripción

A continuación se puede observar en la ***Tabla 4-1*** que se publica el mensaje de activación (paso 7) con su identificador para que la aplicación pueda trabajar con él. En la ***Figura 4-1*** se pueden observar los mensajes 44 y 46 que indican que se ha realizado con éxito dicha publicación.

El mensaje que el dispositivo publica en el tópico "device/activated" con las direcciones de todos los sensores activos se realiza correctamente. La ***Tabla 4-2*** muestra los mensajes de la ejecución de la aplicación, observando en primer lugar como se subscribe al tópico "device/activated" para poder recibir la dirección que el dispositivo a enviado a este tópico.

Tabla 4-2: Pasos 1 y 2 del diagrama

```
1  Subscribing to topic: device/activated
2  Device 30:AE:A4:00:86:24 detected
```

En la ***Figura 4-1*****¡Error! No se encuentra el origen de la referencia.**, que muestra la captura del tráfico, podemos ver como el mensaje es enviado por el dispositivo al gestor de mensajería (mensaje 44) y éste lo envía hacia la aplicación (mensaje 46) Ahora es el turno de la aplicación, ya que sabe los sensores que están activos en el dispositivo.

4.2 Prueba de recepción de muestras

Después de comrobar que los mensajes de activación se han publicado y recibido correctamente, se dará paso al resto de la conexión.

Una vez que la aplicación posee la dirección del dispositivo, se subscribe a los dos tópicos mostrados en la ***Tabla 4-3***, que corresponden con los pasos 9 y 10 del diagrama de la ***Figura 3-2***, para poder recibir las medidas y la configuración de la Pycom Se puede observar que estos tópicos contienen la dirección de la placa para que el intercambio de mensajes se realice de forma organizada, es decir, que como ambos tienen la dirección de la Pycom solo ellos pueden enviarse mensajes entre si.

Tabla 4-3: Pasos 9 y 10 del diagrama

```
1  Subscribing to topic: device/30:AE:A4:00:86:24/sample
2  Subscribing to topic: device/30:AE:A4:00:86:24/config
```

En la ***Figura 4-1*** se puede ver como primero se subscribe a los dos tópicos principales dónde recibirá las medidas y configuraciones provenientes del dispositivo. Como se puede observar los mensajes de subscripción son recibidos correctamente ya que se puede apreciar el mensaje Ack de confirmación, además en la ***Figura 4-3*****¡Error! No se encuentra el origen de la referencia.** se pueden ver en detalle los mensajes 70, 71, 72 y 73.

```
Subscribe Request (id=2) [device/30:AE:A4:00:86:24/sample]
Subscribe Ack (id=2)
Subscribe Request (id=3) [device/30:AE:A4:00:86:24/config], Publish Message [device/30:AE:A4:00:86:24/getsamples]
Subscribe Ack (id=3)
```

Figura 4-3: Detalle de los mensajes 70-73

Una vez que la aplicación está subscrita a todos los tópicos debería poder solicitar y recibir medidas y configuraciones sin problemas. Por ello, el siguiente paso que se tendría que llevar a cabo es la publicación (paso 11 de la ***Figura 3-2***), por parte de la aplicación, de los mensajes solicitando las medidas de cada sensor en el tópico "getSample" ya que en este caso se ejecuta el bloque de muestras.

En la siguiente imagen, ***Tabla 4-4***, se muestra como se envían desde la aplicación los mensajes de solicitud para cada sensor del dispositivo, que se encuentra indicado justo antes de la publicación. Como se puede apreciar, la dirección de cada sensor esta expresado en formato decimal, de tal manera que la dirección del primero es 0x0129 (297d) que corresponde al sensor de luz ambienta, la del segundo 0x0130 (304d) del acelerómetro, la del tercero 0x0140 (320d) del sensore de humedad y temperatura y la del último, sensor de presión y altitud, 0x0160 (352d)

Tabla 4-4: Publicación de Solicitud de Medidas

```
1  SensorId: 297
2  Publishing message to topic: device/30:AE:A4:00:86:24/getsamples
3  SensorId: 304
4  Publishing message to topic: device/30:AE:A4:00:86:24/getsamples
5  SensorId: 320
6  Publishing message to topic: device/30:AE:A4:00:86:24/getsamples
7  SensorId: 352
8  Publishing message to topic: device/30:AE:A4:00:86:24/getsamples
```

Si todo ha ido bien, el dispositivo ha tenido que recibir los mensajes y realizado las medidas en los sensores que se le han especificado. La ***Figura 4-1*****¡Error! No se encuentra el origen de la referencia.** nos indica que la aplicación manda las solicitudes de medidas para los cuatro sensores activos de forma correcta. En la imagen **¡Error! No se encuentra**

el origen de la referencia.*Figura 4-4* se pueden observar en detalle cuatro mensajes (75, 83, 87 y 93) en el tópico "getSample" que corresponden con las peticiones.

```
Publish Message [device/30:AE:A4:00:86:24/getsamples]
Publish Message [device/30:AE:A4:00:86:24/getsamples]
Publish Message [device/30:AE:A4:00:86:24/getsamples], Publish Message [device/30:AE:A4:00:86:24/getsamples]
Publish Message [device/30:AE:A4:00:86:24/sample], Publish Message [device/30:AE:A4:00:86:24/sample]
Publish Message [device/30:AE:A4:00:86:24/getsamples], Publish Message [device/30:AE:A4:00:86:24/getsamples]
```

Figura 4-4: Detalle de los mensajes 75-93

Por último, el dispositivo debe mandar a la aplicación las medidas que ha realizado. Volviendo a la ***Tabla 4-1*****¡Error! No se encuentra el origen de la referencia.** se puede comprobar que los cuatro mensajes con las medidas se han publicado en el tópico "sample" (paso 12 de la ***Figura 3-2***), al cual ya se había subscrito la aplicación.

Se pueden observar en la **¡Error! No se encuentra el origen de la referencia.*Figura 4-1*** y, en detalle en la **¡Error! No se encuentra el origen de la referencia.*Figura 4-5***, que las publicaciones con el tópico "sample" corresponden a los mensajes con las muestras tomadas por los sensores. Vemos que se por cada sensor se observan dos mensajes, el primero que es enviado del dispositivo al gestor de mensajería y el segundo que lo envía el gestor a la aplicación.

```
Publish Message [device/30:AE:A4:00:86:24/sample], Publish Message [device/30:AE:A4:00:86:24/sample]
Publish Message [device/30:AE:A4:00:86:24/getsamples], Publish Message [device/30:AE:A4:00:86:24/getsamples]
Publish Message [device/30:AE:A4:00:86:24/sample]
Publish Message [device/30:AE:A4:00:86:24/sample]
Publish Message [device/30:AE:A4:00:86:24/sample]
Publish Message [device/30:AE:A4:00:86:24/sample]
Publish Message [device/30:AE:A4:00:86:24/sample]
Publish Message [device/30:AE:A4:00:86:24/sample]
```

Figura 4-5: Detalle mensajes 92-146

La aplicación puede obtener los mensajes con las medidas de los sensores que se han publicado en el tópico. Las imágenes que se muestran a continuación muestran los mensajes en la consola que indican que se han recibido los valores correctamente.

Tabla 4-5: Medidas de Luz Ambiental recibidas

```
1  message topic: device/30:AE:A4:00:86:24/sample
2  Received Samples sensorId: 297
3  channelRed: 36
4  channelBlue: 42
```

En la captura anterior ***Tabla 4-5*** se muestra como se ha recibido el mensaje en la aplicación con las medidas del sensor de luz ambiental. La consola nos indica cual ha sido el mensaje recibido y en que tópico se ha publicado. Posteriormente la aplicación procesa el mensaje y se puede ver de forma clara el Id del sensor y una tupla con los valores de la medición. Finalmente, pinta por pantalla los campos de la medición con el valor correspondiente de cada uno de ellos.

El mismo proceso que se ha visto con el sensor de luz ambiental ocurre con el acelerómetro

Tabla 4-6: Medidas del Acelerómetro recibidas

```
1  message topic: device/30:AE:A4:00:86:24/sample'
2  Received Samples sensorId: 304
3  accelerationX: -0.006103515625
4  accelerationY: 0.0811767578125
5  accelerationZ: 1.0062255859375
6  roll: 0.08341002464294434
7  pitch: -4.6122355461120605
```

En la ***Tabla 4-6*** se muestra por pantalla que el mensaje ha sido recibido por la aplicación. También se muestra el Id del sensor y los campos de medición: aceleración en X, aceleración en Y, aceleración en Z, el roll (balanceo) y el pitch (inclinación)

Tabla 4-7: Medidas de Humedad y Temperatura recibidas

```
1  message topic: device/30:AE:A4:00:86:24/sample
2  Received Samples sensorId: 320
3  Humidity: 26.016754150390625
4  Temperature: 30.70318603515625
```

Por otro lado, la aplicación recibe de forma correcta las medidas tomadas por el sensor de humedad y temperatura. En la captura anterior, ***Tabla 4-7***, se pueden ver los valores de los parámetros y la dirección del sensor.

El último mensaje recibido con medidas viene del sensor de presión atmosférica y altitud, que muestra sus valores en la siguiente tabla (***Tabla 4-8***):

Tabla 4-8: Medidas de Presión y Altitud recibidas

```
1  message topic: device/30:AE:A4:00:86:24/sample
2  Received Samples sensorId: 352
3  Pressure: 92964.5
4  Altitude: 719.5625
```

La consola nos muestra todo lo que ha ocurrido mediante mensajes de control durante la ejecución del programa a modo de depuración del programa. Pero para poder comprobar que todos los mensajes se han enviado correctamente y que las publicaciones se han realizado en los tópicos correspondientes se ha utilizado una herramienta que tiene Mosquitto para poder ver las interacciones que ha habido con el gestor de mensajería. Para ello se ha ejecutado el comando de la ***Tabla 4-9*** en la aplicación con el fin de verificar las publicaciones que ha realizado y recibido:

Tabla 4-9: Comando de tráfico de mensajes del broker

```
mosquitto_sub -t device/# -v -h 172.254.1.15
```

A continuación se muestra en pantalla los resultados del comando durante la ejecución de todo el proceso desde el punto de vista de la aplicación (***Tabla 4-10***).

Tabla 4-10: Comprobación con herramienta Mosquitto

```
1  device/activated
2  device/30:AE:A4:00:86:24/getsamples
3  device/30:AE:A4:00:86:24/getsamples
4  device/30:AE:A4:00:86:24/getsamples
5  device/30:AE:A4:00:86:24/getsamples
6  device/30:AE:A4:00:86:24/sample
7  device/30:AE:A4:00:86:24/sample
8  device/30:AE:A4:00:86:24/sample
9  device/30:AE:A4:00:86:24/sample
```

El primer mensaje nos muestra que ha habido una publicación en el tópico "device/activated" que corresponde con el mensaje enviado por el dispositivo en el paso 7 del diagrama, ***Figura 3-2***. Los mensajes publicados en el tópico "getSamples" indican que las peticiones de medición de los cuatro sensores se realizan correctamente, es decir, el paso 10 del diagrama se cumple sin problemas. De la misma manera, se puede ver que la aplicación recibe las medidas por las cuatro publicaciones en el tópico "sample" (paso 12).

Gracias a las capturas obtenidas se puede comprobar y afirmar el correcto funcionamiento de la ejecución completa de los programas lanzados en ambos extremos.

4.3 Prueba de almacenamiento de muestras y visualización de datos

Grafana es una aplicación que permite al usuario conectar una base de datos a su herramienta pata poder mostrar la información de una manera personalizada y dinámica. Además, las gráficas se van actualizando progresivamente con la entrada de nuevas muestras.

En nuestro caso, se ha conectado la herramienta a InluxDB que almacena todos los datos recogidos del dispositivo. En la siguiente imagen, ***Figura 4-6*** se pueden observar el panel con las diferentes representaciones para cada una de las medidas tomadas por nuestro dispositivo.En el panel diseñado, se ha añadido un filtro para poder filtrar los datos por dirección del dispositivo con el fin de poder mostrar los datos correspondientes al dispositivo seleccionado.

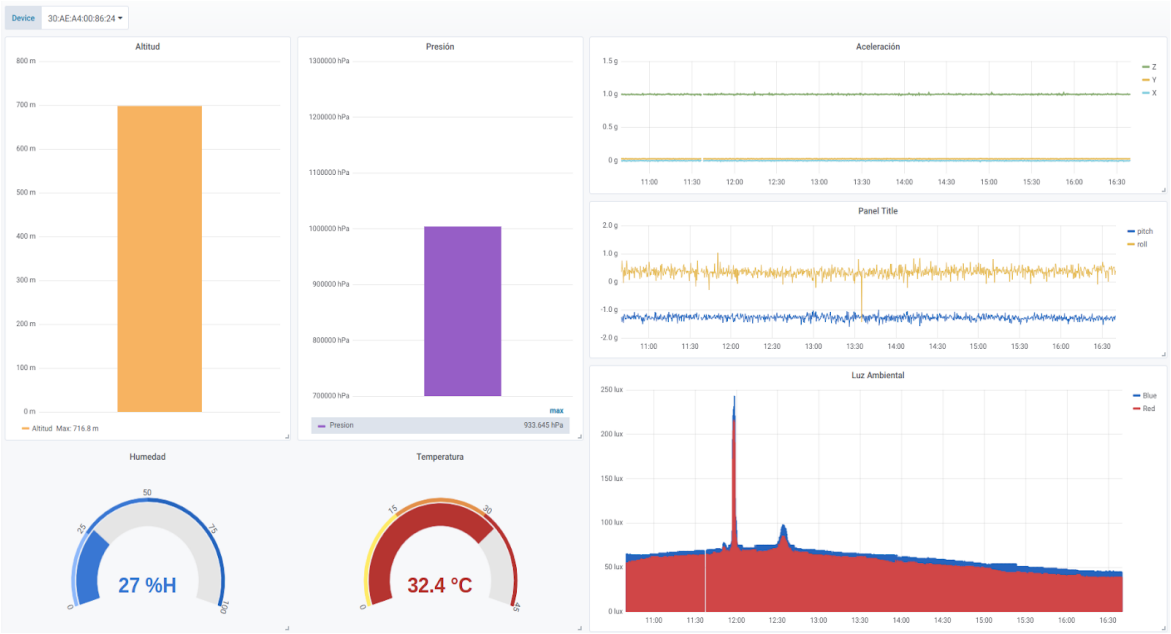

Figura 4-6: Medidas en Grafana

Se distinguen dos histogramas de barras para medir la altitud y la presión. La altitud, representada de color naranja, se mide en metros mientras que la presión, en color violeta, se mide en hectopascales. En la ***Figura 4-7*** se pueden observar con más detalle estos mismos gráficos.

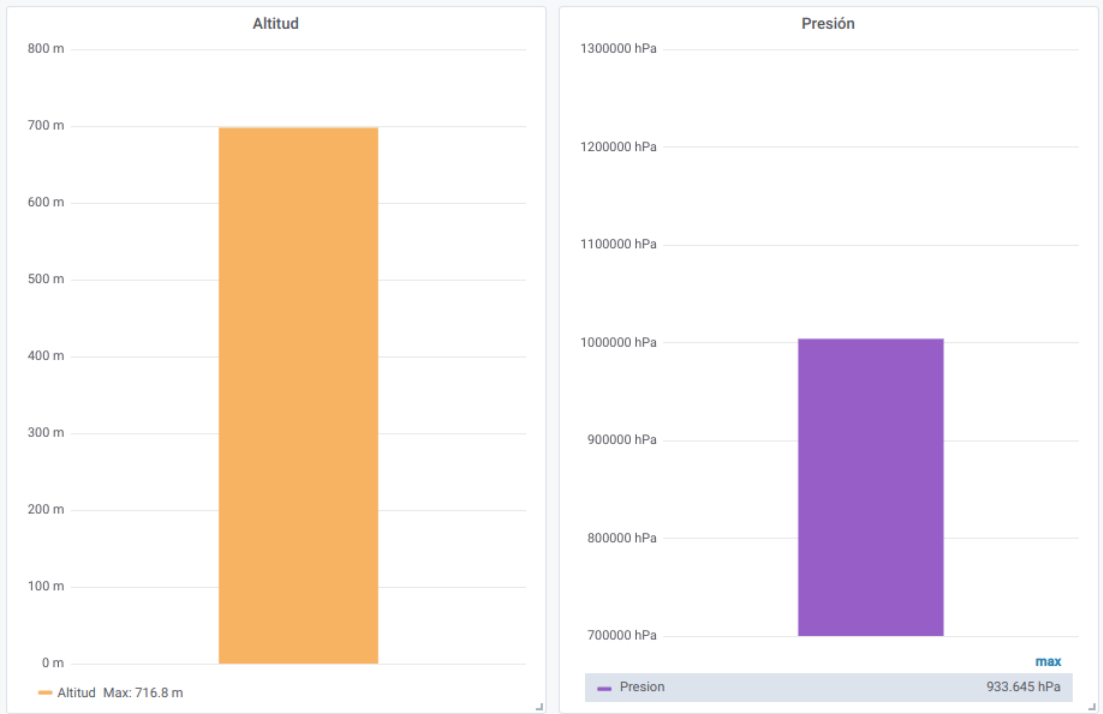

Figura 4-7: Presión y altitud

La humedad relativa y la temperatura se muestran en dos gráficas en el extremo inferior izquierdo de la ***Figura 4-6***; como podemos observar, la temperatura está en grados centígrados y la humedad relativa en porcentaje. De igual manera, en la Figura 4-8 se muestran con más detalle.

Por otro lado, para el sensor de nivel de iluminación, se representan en la gráfica inferior derecha los dos canales de luz ambiental expresados en "lux". La ***Figura 4-9*** muestra una vista ampliada de la representación.

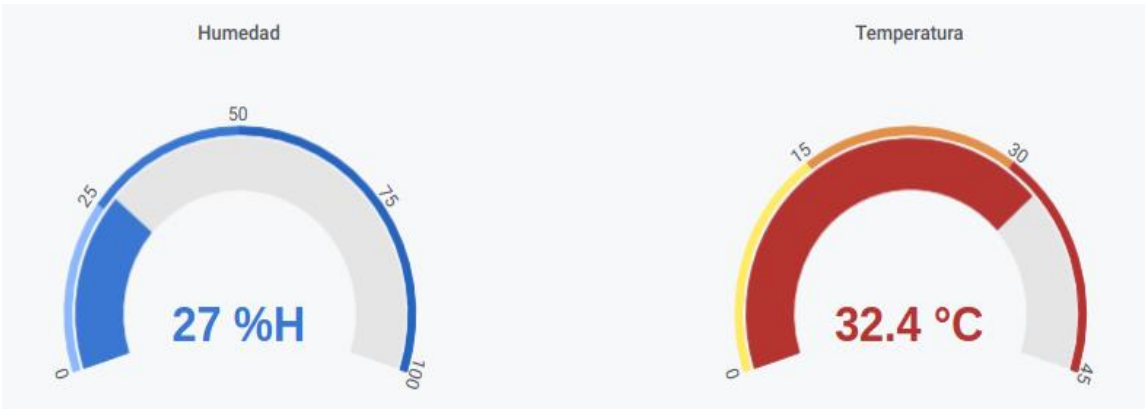

Figura 4-8: Humedad y temperatura

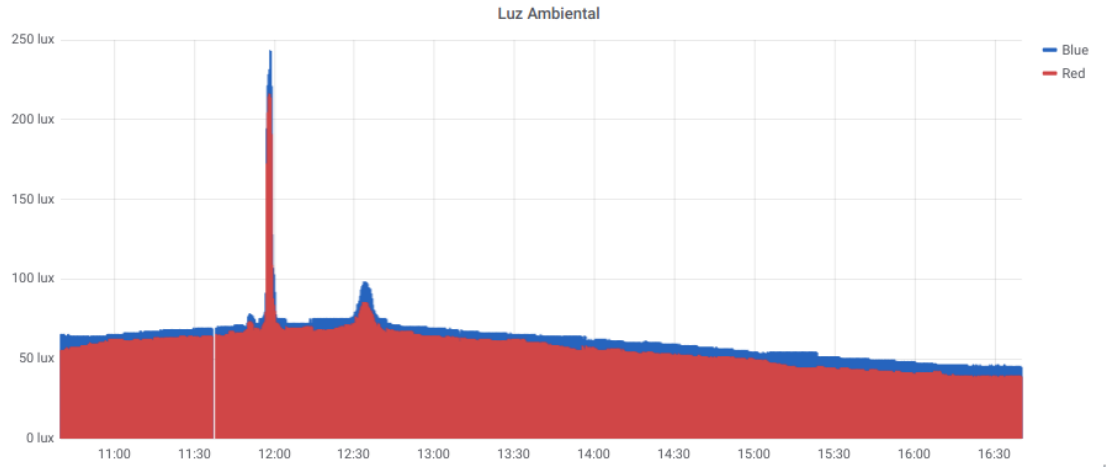

Figura 4-9: Luz ambiental

Por último, los valores tomados por el acelerómetro se muestran en dos gráficas de líneas. Las tres componentes de aceleración por eje (X, Y, Z) se muestran en la ***Figura 4-10*** y están expresadas en Gs. La rotación (roll) y el balanceo (pitch) respecto de la gravedad terrestre, también se muestran en Gs y se detallan en la ***Figura 4-11***.

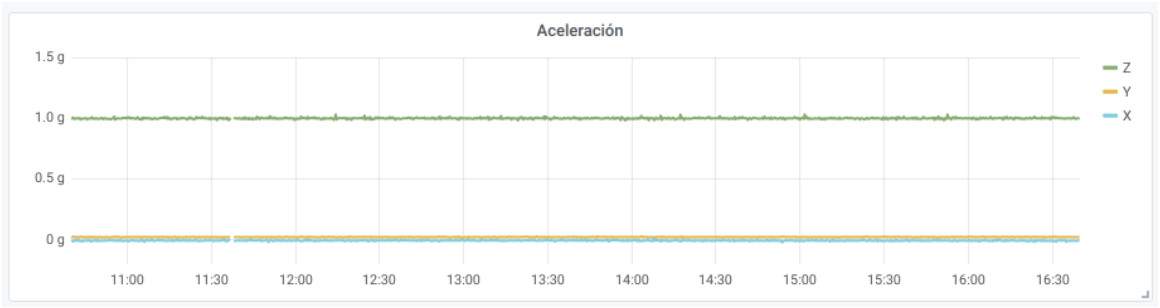

Figura 4-10: Aceleración en cada eje

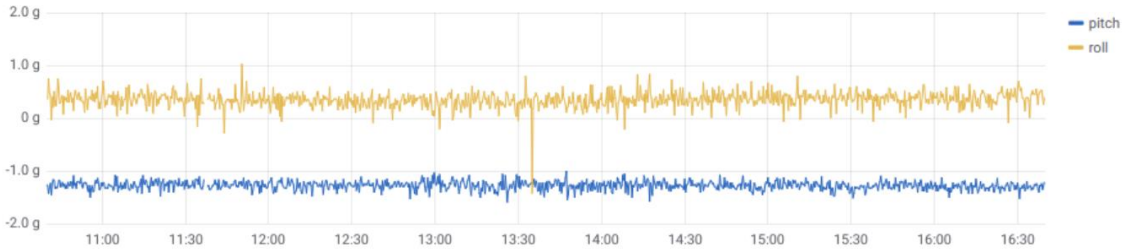

Figura 4-11: Rotación y balanceo del acelerómetro

4.4 Conclusiones

A lo largo de esta sección se han realizado las pruebas necesarias para comprobar que los resultados esperados son correctos, de tal forma que se verifique el funcionamiento del sistema diseñado.

En primer lugar, se ha podido comoprobar gracias a los mensajes de la consola que la ejecución de ambos programas se ha realizado de forma exitosa, de manera que las comunicaciones con el bróker de mensajería se han completado correctamente. También se ha utilizado la herramienta Wireshark para ver el tráfico de la red con el propósito de

observar la mensajería que pasa por el gestor y que es enviada y recibida tanto por la aplicación como por el dispositivo.

Finalmente, el uso de Grafana ha permitido mostrar los valores obtenidos en gráficas personalizadas por el usuario. Estas gráficas muestran las medidas de los sensores de tal forma que se van actualizando según la configuración del usuario.

5 Conclusiones y trabajo futuro

En este capítulo se resumen las conclusiones del trabajo y se detallan posibles líneas de desarrollo futuras relacionadas con este proyecto.

5.1 Conclusiones

El desarrollo de un software para dispositivos IoT, que obtuviera medidas del entorno y, además, permitiera la conexión de dispositivos a la red ha sido realizado de forma exitosa.

Se partía de la base de conseguir un sistema de mensajería que utilizara un protocolo que fuese ideal para los dispositivos que se manejan en el paradigma de IoT. Para ello se ha diseñado una arquitectura acorde con nuestro sistema (aplicación y dispositivo) y se ha empleado un gestor de mensajería o bróker para comunicar los diferentes componentes.

Tanto en la aplicación como en el dispositivo, se ha implementado un software cuya comunicación con el bróker de Mosquitto se establece mediante MQTT. Como se ha explicado en el capítulo 3 el uso de este protocolo es un acierto en cuando a tecnología IoT se refiere, ya que está especialmente diseñado para ser utilizado en dispositivos IoT. El diseño de toda la arquitectura tanto a nivel físico como a nivel de software funciona de manera correcta consiguiendo que la aplicación y el dispositivo puedan intercambiar información a través del gestor de mensajería.

Se han realizado pruebas del funcionamiento del sistema probando todos los elementos del mismo cuyos resultados han sido satisfactorios. Estas pruebas han consistido en realizar pruebas progresivas de los componentes, empezando por la detección de la activación de dispositivos, continuando con el envío de muestras tomadas por los sensores del dispositivo hacia la aplicación a petición de esta última, y el almacenamiento y representación gráfica de dichas muestras. Todas estas pruebas se han respaldado por mensajes de consola y capturas de tráfico hechas con Wirshark.

La aplicación consigue almacenar los datos en InfluxDB, una base de datos de series temporales. Además, se ha empeado la aplicación web Grafana, conectada con InfluxDB para generar paneles de visualización con los datos recogidos de los sensores. Grafana permite generar tablas y gráficos personalizables por el usuario de manera sencilla. Para ello se ha tenido que modificar el código de la aplicación para que conectase con Docker y poder gestionar los datos producidos. Posteriormente, se tuvo que mostrar en el entorno de Grafana gracias a InfluxDB.

5.2 Trabajo futuro

Este trabajo ha servido como punto introductorio a las tecnologías de Internet de las Cosas y su uso en la adquisición de medidas ambientales. Quedan muchos frentes abiertos para mejorar el trabajo y conseguir una aplicación mucho más sólida y efectiva. Por ejemplo, la implementación de métodos para ahorrar energía y trabajar con la mínima posible no ha sido posible por falta de tiempo. De la misma forma, tampoco se ha podido implementar el estado de hibernación de la placa que trataría de permanecer todo el tiempo posible dormida y solo despertarse para realizar medidas. Pero de forma genérica se han cumplido todos los objetivos importantes del proyecto.

En primer lugar, se podría combinar el sistema de recolección de medidas con una plataforma que actúe sobre el entorno en base a las medidas recibidas. Por ejemplo, se

podría implementar un algoritmo en la aplicación en la nube que decidiera cuándo regar un campo y mandara comandos de activación a un sistema de riego controlable remotamente

Por otro lado, implementar algún sistema de ahorro de energía empleando funciones de "sueño profundo" para los dispositivos IoT de forma que cuando no tengan nada que hacer, en vez de paralizar el procesador y seguir consumiendo energía, se podría preconfigurar un temporizador y apagar el procesador pasando el dispositivo a modo de ultra bajo consumo de energía hasta que expire el temporizador, momento en que se despertaría el dispositivo de nuevo. De esta forma los dispositivos que recogen las medidas de su entorno consumirán muy poca energía y solo estarían activos el tiempo estrictamente necesario.

Finalmente, se podría incorporar al sistema funcionalidades de reconfiguración de los dispositivos permitiendo ajustar parámetros como la periodicidad con la que el dispositivo se despierta, el tiempo que permanece despierto, etc.

Referencias

[1] ¿Qué es Internet de las cosas (IoT)? - Definición en WhatIs.com. 16 junio, 2019, https://searchdatacenter.techtarget.com/es/definicion/Internet-de-las-cosas-IoT

[2] The Paho MQTT Python Client-Beginners Guide. (2019, 12 junio). 16 junio, 2019, de http://www.steves-internet-guide.com/into-mqtt-python-client/

[3] What is a Container? | Docker. 16 junio, 2019, de https://www.docker.com

[4] About InfluxData: Who We Are | InfluxData. 16 junio, 2019, de https://www.influxdata.com

[5] Grafana - The open platform for analytics and monitoring. 16 junio, 2019, de https://grafana.com/

[6] Dispositivos IoT: Aplicaciones y Ejemplos ~ IoT World Online. (2018, 15 julio). 16 junio, 2019, de https://www.iotworldonline.es/dispositivos-iot-aplicaciones-y-ejemplos/

[7] MicroPython. (2019, 9 junio). [lenguaje de programación]. 16 junio, 2019, de https://micropython.org

[8] Diferentes protocolos para micros. (2018, 28 junio). 17 julio, 2019, de https://www.drouiz.com/blog/2018/06/25/uart-vs-spi-vs-i2c-diferencias-entre-protocolos/

[9] WiPy 3.0 - Pycom. 16 junio, 2019, de https://pycom.io/product/wipy-3-0/

[10] Espressif Esp32 Datasheet . 16 junio, 2019, de https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[11] Pysense - Pycom. 16 junio, 2019, de https://pycom.io/product/pysense/

[12] Ambient Light Sensor. 16 junio, 2019, de https://www.mouser.com/ds/2/239/Lite-On_LTR-329ALS-01%20DS_ver1.1-348647.pdf

[13] Accelerometer. 16 junio, 2019, de:https://www.st.com/resource/en/datasheet/lis2hh12.pdf

[14] Humidity and Temperature Sensor. 16 junio, 2019, de https://www.silabs.com/documents/public/data-sheets/Si7006-A20.pdf

[15] Pressure and Altitude Sensor: https://www.nxp.com/docs/en/data-sheet/MPL3115A2.pdf

[16] Eclipse Paho - MQTT and MQTT-SN software. 11 julio, 2019, de http://www.eclipse.org/paho/

[17] MQTT · GitBook. 16 junio, 2019, de https://docs.pycom.io/tutorials/all/mqtt.html

[18] Protocolo http. 24 junio, 2019, de http://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/http.html

[19] MQTT VS HTTP: IBM Bluemix and the Internet of Things - Workshop. (2015, 28 abril). 16 junio, 2019, de https://www.slideshare.net/gjuljo/ibm-bluemix-and-the-internet-of-things-workshop

[20] Python Documentation contents — Python 2.7.16 documentation. 16 junio, 2019, de https://python.org.html

[21] Machine · GitBook. 16 junio, 2019, de https://docs.pycom.io/firmwareapi/pycom/machine/

[22] Pysense · GitBook. 16 junio, 2019, de https://docs.pycom.io/pytrackpysense/apireference/pysense.html

[23] 5. Data Structures — Python 3.7.3 documentation. 16 junio, 2019, de https://docs.python.org/3/tutorial/datastructures.html

[24] Stack Overflow - Where Developers Learn, Share, & Build Careers. 16 junio, 2019, de https://stackoverflow.com/

Glosario

ADC	Analogic Digital Converter
API	Application Programming Interface
BBDD	Bases de Datos
GPIO	General Purpose Input/Output
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
I2C	Inter-Integrated Circuit
MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport
PWM	Pulse Width Modulation
QoS	Quality of Service
RAM	Random Access Memory
SoC	System on Chip
SPI	Serial Peripheral Interface
SQL	Structured Query Language

Anexos

A. Manual de instalación

La librería que implementa el protocolo MQTT deberá ser instalado en la Raspberry. Se deberá instalar Paho con el siguiente comando desde un terminal de Linux:

```
sudo pip install paho-mqtt
```

Para depurar correctamente el código y comprobar errores como tabulaciones o detectar donde se produce alguna implementación innecesaria de código se ha utilizado "pylint":

```
sudo pip install pylint
```

La herramienta InlfuxDB también necesitó ser utilizada en nuestra aplicación. El comando a ejecutar en la Raspberry para su instalación es el siguiente:

```
sudo pip install influxdb
```

B. Manual del programador

B.1 Código Explicado

En este anexo se va mostrar algunas de las explicaciones más relevantes en el funcionamiento de varios extractos de código. [23][24]

Dentro del archivo "ApplicationDefinitios.py" se pueden observar los siguientes métodos:

def on_device_activated

El pseudocódigo que se muestra en el cuadro inferior muestra como la aplicación procesa el mensaje de activación del dispositivo. De esta manera extrae los campos necesarios para operaciones futuras en la ejecución. También se puede ver como se realizan las dos primeras subscripciones.

```
header = struct.unpack(formato, mensaje)
devId = ':'.join(map(lambda b: '%02X' % b, header[0:6]))
numSensorIds = header[6]
sensorIds = []
msgPos = SIZE_MSG_FORMAT_ACTIVATED

print(tópico)
client.subscribe(tópico)
print tópico)
client.subscribe(tópico)
```

En el siguiente cuadro de texto se muestra la siguiente parte de este método. El bucle recorre la lista de sensores para extraer la dirección del mensaje y publicarla para la solicitud de medidas del propio sensor. Por tanto, este bucle se realizará para cada sensor activo.

```
for _ in range(numSensorIds):

     sensorId, = struct.unpack(formato, mensaje)
     print(sensorId)
     msgPos += SIZE_MSG_FORMAT_SENSORID

     print(tópico)
     getSamplesMsg = struct.pack(formato, mensaje)
     client.publish(tópico, mensaje)
```

def on_device_sample (client, devId, message)

En este método se extraen las medidas de cada sensor y se conecta a un cliente de InfluxDB que nos permitirá trabajar en Grafana con los datos obtenidos. Una vez que se ha conectado con Influx se procede a crear las tablas para cada tipo de sensor:

```
global sampleCounter

     sensorId, = struct.unpack(formato, mensaje)
     values = unpack_sample(message, sensorId)

     print(sensorId)
     print(values)

     sampleTime = datetime.datetime.utcnow().isoformat()

     client = influxdb.client.InfluxDBClient(host, port)
     databaseNames = map(name, client.get_list_database())

     if(INFLUX_DATABASE no existe):
           client.create_database(INFLUX_DATABASE)

     client.switch_database(INFLUX_DATABASE)

     if (sensorId == SENSORID_AMBIENT_LIGHT):
          print(valores)
          client.write_points([{
               'measurement': INFLUX_TABLE,
               'time':tiempo
               'tags': {devId, sensorId,},
               'fields' : {
                    'channelRed': values[0],
                    'channelBlue': values[1]}
          }])
          client.close()
```

Esta última condición se repetirá para cada uno de los sensores que estén activos en el dispositivo con sus respectivos campos de valores devueltos.

def unpack_sample (msg)

El siguiente recuadro muestra como se realiza una comprobación del mensaje con las medidas de cada sensor y se procede a desempaquetar los valores del mensaje.

```
if sensorId == SENSORID_AMBIENT_LIGHT:
     valores, = struct.unpack(formato, mensaje)
     return valores

     elif sensorId == SENSORID_ACCELERATION:
```

De igual forma que en el caso anterior se debe realizar esta condición para cada uno de los sensores del dispositivo.

Por otro lado, el estudio del código desarrollado para el dispositivo [5] es necesario para el entendimiento de su funcionamiento. A continuación se muestran algunos de los métodos más relevantes:

def publish_activated(client)

En este método recorremos la lista de sensores activos y mediante un bucle introducimos las direcciones de estos sensores en un mensaje. Finalmente se realiza la publicación de dicho mensaje:

```
msgActivated = struct.pack(formato, sensorlist, tamaño lista)

     for sensorId in sensorIds:
          msgActivated += struct.pack(formato, sensorId)

     client.publish(tópico, mensaje)
```

def AmbientLight(sensorId)

Se usa este método para acceder a la librería del sensor de luz ambiental y tomar los valores resultantes de la medida. Posteriormente se empaqueta el mensaje que contiene el identificador del sensor y las medidas tomadas:

```
AmbL = LTR329ALS01()
     Light = AmbL.light()
     ALValues = struct.pack(formato, sensorId, valores)

     return ALValues
```

La metodología para los demás sensores es la misma, por lo que los métodos serán acordes con cada sensor y los valores que devuelve cada uno.

B.2 Añadir Sensor

Cada dispositivo IoT es diferente, por lo que los sensores con los que cuenta cada uno varían. Por ello es necesario explicar cómo añadir un nuevo sensor al software del dispositivo.

En primer lugar hay que añadir una variable en el archivo “Common.py” que contenga la dirección del nuevo sensor.

En la aplicación se debe modificar el archivo “ApplicationDefinitions.py” de tal forma que se añadan los campos del nuevo sensor en la conexión con InfluxDB. Además, de crear una condición nueva para desempaquetar los mensajes con las muestras que se reciban.

En el dispositivo deberá modificarse el software “DeviceDefinitions.py”. Deberá añadirse las librerías del nuevo sensor y añadir este mismo a la lista de sensores disponibles. También será necesario crear un método que acceda a la librería del nuevo sensor para que se active y tome las medidas propias del mismo. Finalmente se añadirá la

nueva condición para empaquetar los mensajes que contienen las muestras de este nuevo sensor.

B.3 Conectar y configurar InfluxDB

InlfuxDb es una herramienta que nos va a permitir mostrar los resultados obtenidos de la ejecución con Grafana. Para ello se debe establecer una conexión con InfluxDB donde se almacenen los datos obtenidos por los sensores.

Para configurar InfluxDB en el software se añade la librería que nos facilita InfluxDB y se ejecutan las siguientes líneas de código:

```
client = influxdb.client.InfluxDBClient(host=INFLUX_SERVER, port=INFLUX_PORT)
databaseNames = map(lambda name: name['name'], client.get_list_database())

if(INFLUX_DB not in databaseNames): client.create_database(INFLUX_DB)
        client.switch_database(INFLUX_DB)
```

El servidor de InfluxDB tiene la dirección IP 127.0.0.1 y escucha en el puerto 8086. La variable INFLUX_DB hace referencia al nombre que se quiere poner a la base de datos que se va a generar.

B.4 Configurar gráficos en Grafana

La herramienta Grafana se ha utilizado para mostrar los datos obtenidos de las mediciones de los sensores en tablas y gráficos dinámicos personalizados por el usuario. Para su uso se ha necesitado la ayuda de InfluxDB y Docker.

El primer paso para crear un cuadro de mandos en Grafana es configurar esta aplicación para que conecte con InfluxDB y pueda acceder a las series de datos temporales. Para ello se debe seleccionar la fuente de datos, en este caso InfluxDB, y añadir la dirección IP del contenedor con toda la información.

Una vez cargada la base de datos se pueden crear tablas y gráficas mediante un formato de “query” que nos proporciona Grafana. Primer debemos añadir gráfico, lo que nos llevará a un menú donde debemos introducir los campos que queremos que muestre; así como los filtros y condiciones que queramos aplicar. Una vez creada la tabla con los datos que nos va mostrar la tabla, se pasa a modificar el aspecto de la tabla mediante otro menú que permite personalizar dicha tabla. Esto también nos permite ajustar las unidades de los datos y acotar las medidas en los rangos que quiera el usuario.

Una vez añadidos todos los gráficos que se deseen se puede modificar tanto el tamaño como la disposición de los mismos.

Por otro lado, Grafana también permite aplicar variables o filtros mediante los cuales se pueden mostrar los diferentes cuadros creados. En nuestro caso, por ejemplo, se ha añadido una variable que filtra los dispositivos disponibles haciendo posible seleccionar uno de ellos y acceder a su propio cuadro de datos.

B.5 Variables: Formatos

En la transmisión de los mensajes se trabaja con multitud de tipos y secuencias de datos. Para cada sensor se necesitará un formato que codifique el mensajes enviado con todos los datos que contendrá el mensaje. En la siguiente tabla se muestran los diferentes formatos de los datos indicando el tipo en lenguaje de programación C y Python, además del tamaño que ocupa cada uno.

Format	C Type	Python type	Standard size
x	pad byte	no value	
c	char	string of length 1	1
b	signed char	integer	1
B	unsigned char	integer	1
?	_Bool	bool	1
h	short	integer	2
H	unsigned short	integer	2
i	int	integer	4
I	unsigned int	integer	4
l	long	integer	4
L	unsigned long	integer	4
q	long long	integer	8
Q	unsigned long long	integer	8
f	float	float	4
d	double	float	8
s	char[]	string	
p	char[]	string	

Tabla B5-1: Estructuras de Datos

Una vez conocidos todos los formatos en los cuales puede aparecer representado un datos, se mostrarán las variables definidas con los formatos de datos para cada tipo de mensaje enviado o procesado.

```
- MSG_FORMAT_GETSAMPLES_MSG_FORMAT      = '!HH'

- MSG_FORMAT_GETSAMPLES_AMB_LIGHT       = '!HHH'
- MSG_FORMAT_GETSAMPLES_ACCELEROMETER   ='!Hfffff'
- MSG_FORMAT_GETSAMPLES_HUM_TEMP        = '!Hff'
- MSG_FORMAT_GETSAMPLES_PRESS_ALT       = '!Hff'
```

```
-  MSG_FORMAT_SAMPLES_AMB_LIGHT          = '!HH'
-  MSG_FORMAT_SAMPLES_ACCELEROMETER      = '!fffff'
-  MSG_FORMAT_SAMPLES_HUM_TEMP           = '!ff'
-  MSG_FORMAT_SAMPLES_PRESS_ALT          = '!ff'

-  MSG_FORMAT_SAMPLES                    = '!Hi'
   MSG_FORMAT_CONFIG_SETTIMER           = '!HI'
-  MSG_FORMAT_CONFIG_SETDEVICEID         = '!HBBBBBB'

-  MAC_FORMAT                            = '!BBBBBB'
-  MSG_FORMAT_ACTIVATED                  = '!BBBBBBB'
-  MSG_FORMAT_SENSORID                   = '!H'
```